# Basic SSH Security

SSH is a powerful command line remote administration utility. It is also used to create secure tunnels for file transfer (scp, sftp), GUI remote administration, and many other tasks. Since SSH is powerful, it is often attacked. Any public IP address open to SSH on TCP port 22 will almost certainly scanned and attacked with basic brute force login attempts. Current scan data is available at <https://isc.sans.edu/port.html?port=22> and top passwords currently attempted are available here: <https://isc.sans.edu/ssh.html>. Please visit both those sites.

One method in use today to reduce exposure to SSH brute force password attempts is to change the port number to something different than 22. While this does help, it is more akin to “security through obscurity” than true security. Another method is to require that users create long, difficult passwords. This works, but there always seem to be users that manage to evade requirements.

An effective way to secure SSH is to use public/private key pairs instead of usernames and passwords. The administrator of a machine using SSH only allows login by those users whose public key is saved on the server. The users authenticate with their private key, which they always keep secure. We will cover the cryptography of public and private key pairs in a later class.

In key-based authentication, someone must generate the key pair the user will use. If the users control the server running the SSH daemon, they will usually generate the key pair themselves. After generating the key pair, they save the private key (securely) on their client workstations and add the public key to the authorized\_keys file in the user’s .ssh directory on the server. Some organizations may generate the key pair for the users, distribute the private key to the users and save the public key on the server. Larger organizations will have Certificate Authorities (CA) and a Public Key Infrastructure (PKI) to distribute and control certificates and keys.

Linux and most cloud services have SSH and the necessary tools built in, so we will discuss them first. Windows 10 ver. 1803 (April 2018 update) and later now include openssh by default.

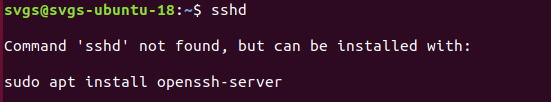
Please read this RedHat description of the SSH protocol: <https://access.redhat.com/documentation/en-us/red_hat_enterprise_linux/6/html/deployment_guide/ch-openssh#s1-ssh-protocol>. Note: On Linux, especially in configuration files, ssh refers to an SSH client and sshd refers to the SSH daemon, or server.

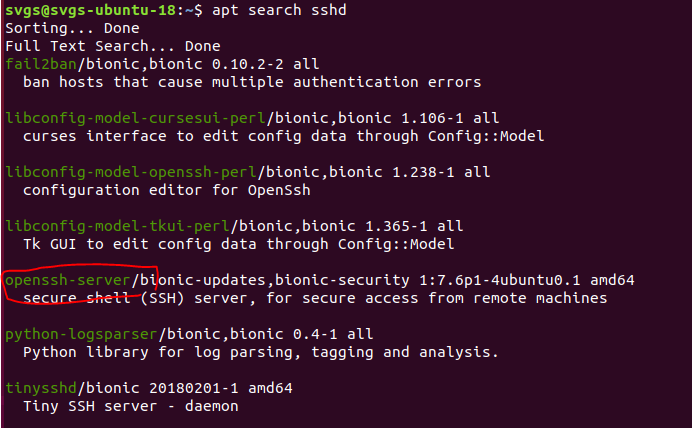
# SSH Lab

For this lab, we will practice SSH connections from our Windows host or VM (SSH client) to our Ubuntu VM (SSH server.) We will create the key pair on the Windows host or VM and then copy the public key to the Ubuntu VM, but it could just as easily be done the other way around. We will perform these steps:

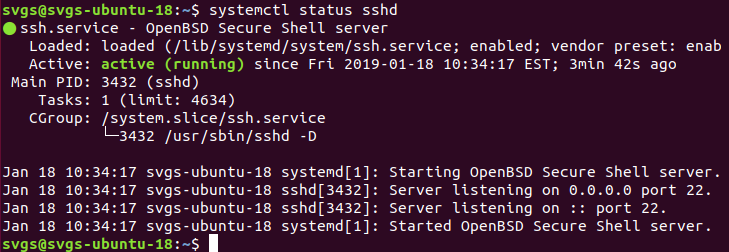
* Test SSH from the Windows host or VM to the Ubuntu VM using a username/password from the Ubuntu VM
* Create a public/private key pair on the Windows host or VM
* Append the public key in the .ssh/authorized\_keys file of the Ubuntu user.
* Test SSH from the Windows host or VM to the Ubuntu VM using the key pair
* Disable username/password authentication for SSH on the Ubuntu VM.

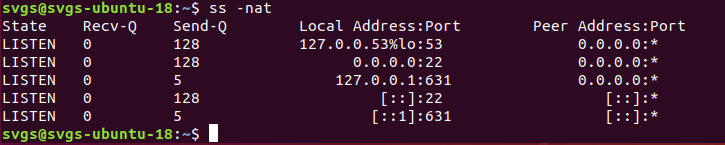
## Install SSH server (sshd) on the Ubuntu VM

Ubuntu server includes sshd by default, but the desktop version we are using does not. We can find the name of the software to install in a couple of ways. If we try to execute sshd, Ubuntu helpfully tells us how to install it.  
(Note: If you installed ssh in an earlier lab and disabled it, you can re-enable it using  
systemctl enable ssh)  


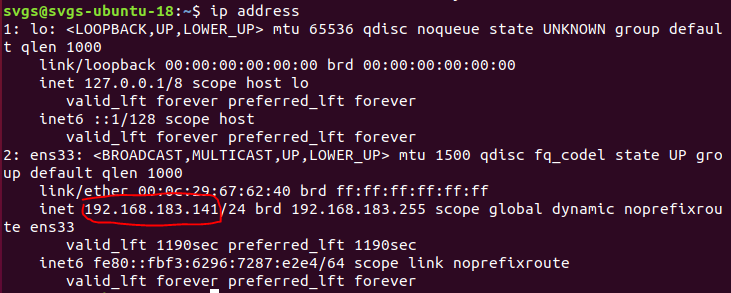
The more orthodox method is to search using apt.  


Install sshd.  
sudo apt-get install openssh-server

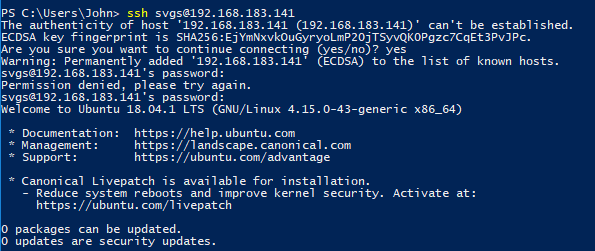
Check that sshd is running with systemctl status sshd  


We can verify that the server is listening on TCP port 22 with ss -nat (ss is the replacement for netstat.)  


## SSH from your Windows host or VM to your Ubuntu VM with username and password

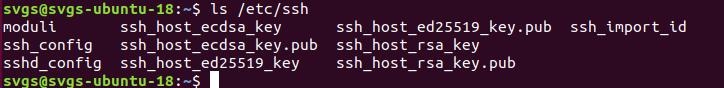
Make sure that your VMs are on the same VMware host network, that their IP addresses are on the same subnet, and that they can ping each other. You can use the command ip address to verify the IP address on Linux and the command ipconfig on Windows

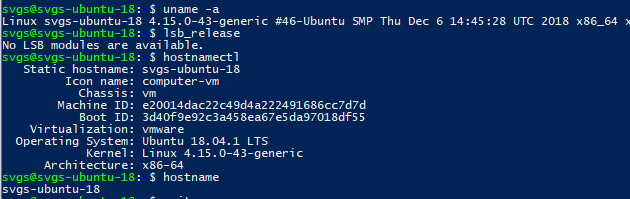
From your Windows host or VM, use ssh --help or man ssh study the syntax. Then create an SSH connection to the Ubuntu VM. The username should be that of a user that exists on the Ubuntu VM.



The SSH server gives its public key to the SSH client so the client knows that it is connecting to the correct server. The SSH client verifies the server’s public key by checking to see if it is listed in the file ~/.ssh/known\_hosts or /etc/ssh/known\_hosts in Linux or in C:\Users\[usesrname]\.ssh\known\_hosts in Windows 10. If the key is not in the list, the client gives you a warning that the authenticity of the server cannot be established. If you are certain that you are connected to the correct server, answer yes and the key will be added to the known\_hosts file.

Another method of verifying the host server (the first time) would be for the owner of the server to make the fingerprints of the public keys available so you can check the fingerprint against the one in the warning. Or, they can make the public keys available so that you can add them to the known\_hosts file yourself. (The server key pairs are kept in /etc/ssh. The ones ending in .pub are the public keys.)



Once you have connected, experiment with some commands. The results should be the same as if you typed them from the Ubuntu console. (In the screenshot, uname -a, lsb\_release, hostnamectl, and hostname are commands that can show information about the Linux host. This just demonstrates that we really are executing commands on the Ubuntu VM. 

When you are done, close the connection.   


## Create a user key pair on the Windows host or VM

On the Windows host or VM, create a public/private key pair for your user with the procedure shown here: <https://access.redhat.com/documentation/en-us/red_hat_enterprise_linux/6/html/deployment_guide/s2-ssh-configuration-keypairs>. Do not create DSA or SSH v1 keys as they are not secure.

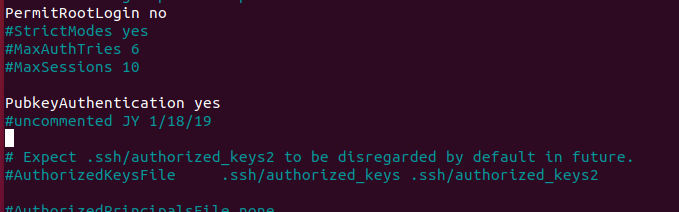
Important points to note:

* There are two versions of SSH, v1 and v2. Always use SSH v2. (Here’s a video of Trinity shutting down a power grid using Nmap and an SSH v1 exploit in The Matrix Reloaded: <https://www.youtube.com/watch?v=0PxTAn4g20U>)
* Don’t use -t dsa or -t rsa1, as DSA (-t dsa) keys have been deprecated and -t rsa1 generates SSH v1 keys. More secure choices are -t ecdsa and -t ed25519, which provide. You can use -t rsa, but it is less secure and may be deprecated in the future; if the server only supports RSA keys, you’ll have to use this however.
* The standard storage location for user SSH keys in Linix is ~/.ssh/
* The .ssh directory should always be rwx------, full rights for the user and no rights for anyone else. That’s what the step chmod 700 ~/.ssh did. If someone else could read your keys, they could impersonate you.
* The passphrase you enter during key generation protects the private key so that only you can use it. Other than allowing the key to be used, it does not affect authentication.
* The root user should never be allowed to login via SSH; always log in with a standard user and use su or sudo when root privileges are needed.

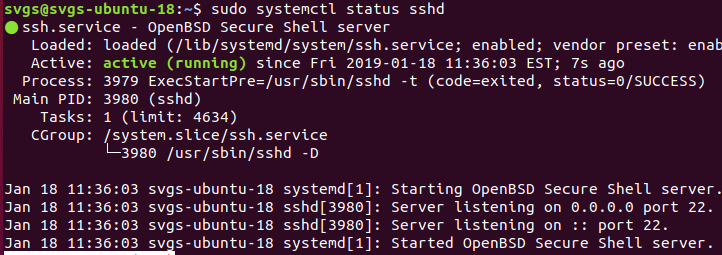
## Configuring the SSH Server for public keys

The SSH daemon, sshd, on the Ubuntu VM is configured using the file, /etc/ssh/sshd\_config. The easiest way to edit the file is to use nano.



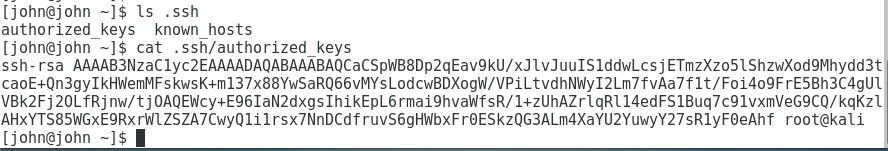
To allow sshd to authenticate public keys, we need to change one line in sshd\_config. The comment in front of “PubkeyAuthentication yes” needs to be removed. Also note that the AuthorizedKeysFile setting will cause the SSH daemon to look for public keys in the users’ .ssh/authorized\_keys file. It is a good idea to add comments of your own whenever you change configuration files so you can trace the changes later. Also edit PermitRootLogins to change it to no.   


You can see explanations and defaults for the sshd\_config parameters by running man sshd\_config

Once you have saved the changes to sshd\_config, you will need to restart sshd.   


Copy the public key to the server

Once sshd is configured, you still need to enter a copy of the user’s public key in the user’s ~/.ssh/authorized\_keys file. One way to do that from a Linux host is to use ssh-copy-id. It explained in Option 1 of this link: [https://www.digitalocean.com/community/tutorials/initial-server-setup-with-Ubuntu-7](https://www.digitalocean.com/community/tutorials/initial-server-setup-with-centos-7). Unfortunately, ssh-copy-id is not included with Windows so we will manually create and configure the authorized\_keys file as in Option 2 of the link.

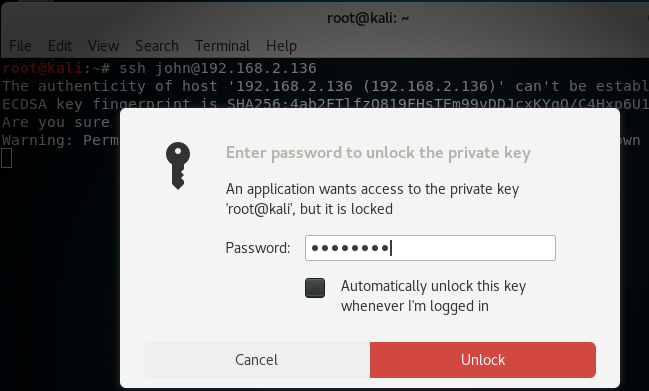
The key is now on the Ubuntu VM.  
 

Note: On servers with many users, the sysadmin may want tighter control over the keys that allow SSH access. In that case, they will prevent sshd from using keys in users’ .ssh directories by setting the AuthorizedKeysFile parameter in sshd\_config to none. Then, sshd will only look for public keys in /etc/ssh/authorized\_keys, which requires root privileges to access. Also, the sysadmin can create a group (allowssh, for example) and set the AllowGroups parameter to the group name.

## SSH to the server using a public key (Finally)

Now, when you SSH from the Windows host or VM to the Ubuntu VM, you should be authenticated automatically. If you did not put a password on your private key, you will see this: 

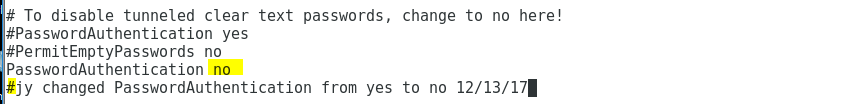
The SSH client on the Windows host or VM used the private key in the .ssh directory to authenticate, and sshd on the Ubuntu VM accepted the authentication because it has the public key.

If your private key is protected with a password, you will see this:  
 

Once you enter the password to unlock the private key, you should immediately be logged in to the Ubuntu VM through SSH.

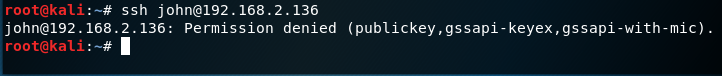
## Disable Password Authentication

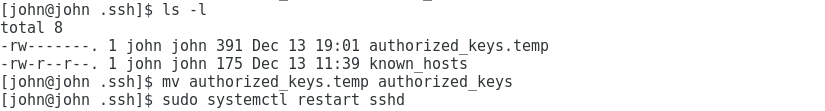
Finally, we need to disable password authentication so that only public key authentication is allowed. We need to edit /etc/ssh/sshd\_config on the Ubuntu VM to change the PasswordAuthentication parameter to no. Again, commenting changes to configuration files is a good thing.



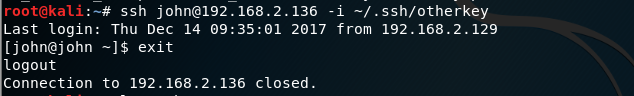
Be sure to save your changes to the sshd\_config file, and then restart the sshd daemon.

We can test this change by temporarily changing the name of the authorized\_keys file in our user’s .ssh directory on the Ubuntu VM. Since the sshd daemon will not find an authorized key, it should deny authentication and not allow us to authenticate with a password.

It worked! 

Let’s change the authorized\_keys.temp file back to its correct name so SSH works again.  

## Multiple Keys

You may have access to several servers, each with its own key. In that case, you can specify which key the SSH client should use with the -i (identity) switch. 

# Windows SSH Clients

The most popular SSH Client for Windows is Putty, available at <https://www.chiark.greenend.org.uk/~sgtatham/putty/latest.html>. However, openssh is now available in Windows 10 ver. 1803 (April 2018 update) and later in PowerShell. It works just like the Linux version.  
