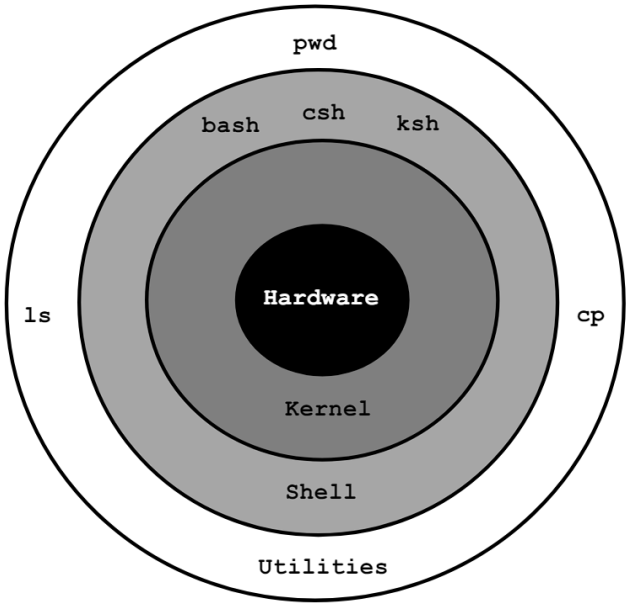
Configure and secure SSH

Linux #redhat #ssh

What exactly is SSH?

- SSH stands for Secure Shell
  - What is a shell?
    - A shell provides you with an interface to the Linux system. It takes in your commands and translate them to kernel to manage hardware.
    - When you actually log into your Linux machine and you get a prompt, and you type in the commands for those prompts, that environment is a shell. It's actually a platform that's given to you to control or give commands to your kernel.
      - When a user logs in, they get a dollar sign ($)
      - When a root logs in, they get a hash sign or a pound sign (#)



- Utilities (`ls`, `pwd`, `cp`) (Controls Shell)
  - Shell (`bash`, `csh`, `ksh`) (Controls Kernel)
    - Kernel (Controls hardware)
      - Hardware (Dell, HP, Apple, etc)
- Open SSH is a package/software
  - By default when you install Linux or Red Hat, that OpenSSH package comes installed on your system.
- When we run OpenSSH, it starts as daemon and that daemon name is 'sshd'.
  - You could stop and restart its daemon if you know the name of the daemon.
- Which port that SSH service runs on?
  - By default the SSH port runs on port number 22
  - Many programs in Linux have a dedicated port numbers that by default, they're run on.

- SSH itself is secure, meaning communication through SSH is always encrypted, but there should be some additional configuration can be done to make it more secure.
- Following are the most common configuration an administrator should take to secure SSH

Configuring Idle Timeout Interval

- Avoid having an unattended SSH session, you can set an Idle timeout interval
  - Become root
  - Edit your `/etc/ssh/sshd_config` file and add the following line:

```
clientAliveInterval 600
clientAliveCountMax 0
```

- - Then restart the daemon using the `systemctl restart sshd`
- The idle timeout interval you are setting is in seconds (600 secs = 10 minutes). Once the interval has passed, the idle user will be automatically logged out
- It's always recommended to make a backup copy of the sshd_config file or any file in /etc for that matter.

Example using `cp` command:

```
[root@localhost ~]# cp /etc/ssh/sshd_config /etc/ssh/sshd_config-orig
```

- Doing a backup of the `/etc/ssh/sshd_config` file by copying the file to another directory or in this case by changing the name of the copy and keeping it in the same directory (`/etc/ssh/`).
- Note we are logged in as root user.

Example using `vi` command:

```
[root@localhost ~]# vi /etc/ssh/sshd_config
```

- Open and edit the `/etc/ssh/sshd_config` file using `vi` file editor.

File editor:

```
...
# override default of no subsystems
Subsystem       sftp    /usr/libexec/openssh/sftp-server

# Example of overriding settings on a per-user basis
#Match User anoncvs
#       X11Forwarding no
#       AllowTcpForwarding no
#       PermitTTY no
#       ForceCommand cvs server
```

- Is best to add the lines that you want to add towards the end of the file because they are not already included in the file and in that way you can now what you added later.
  - If you want to go to the end of the file quickly, type Shift + G.

- Hit O in your keyboard to start a new line.
  - Note when typing O, the INSERT mode is automatically set.
- Then you can type the instructions and remember that any line starting with # will be ignored (they are comments)
  - Note when a valid instruction is typed is it will turn yellow.

Example using `systemctl` command:

```
[root@localhost ~]# systemctl restart sshd
```

- Restart sshd daemon using the `systemctl` command

Example using `cp` command:

```
[root@localhost ~]# cp /etc/ssh/sshd_config-orig /etc/ssh/sshd_config
```

- Bringing back the backup copy you made in case you do not want changes to be done or you just want to restore the old configuration.

Output:

```
cp: overwrite '/etc/ssh/sshd_config'? y
```

- As you are overwriting a file and potentially loosing information previously written in that file it will ask the user to confirm the action
  - Enter "y" for yes and "n" for no

Disable root login

- Disabling root login should be one of the measures you should take when setting up the system for the first time. It disable any user to login to the system with root account
  - Become root
  - Edit your `/etc/ssh/sshd_config` file and replace PermitRootLogin yes to no:
    - For this you can use the `vi` file editor and simply type `/` while not in INSERT mode to look for the term "PermitRootLogin" in the whole file.
    - After finding where it is you can uncomment the line in case of needing it and replace the "yes" keyword to "no".

```
PermitRootLogin no
```

- - Then restart the sshd daemon by running `systemctl restart sshd`

Disable empty Passwords

- You need to prevent remote logins from accounts with empty passwords for added security.
- So any user that does not have a password, they should be limited, they should be forced to not log in unless they create a password.
  - Become root

- Edit your `/etc/ssh/sshd_config` file and remove # from the following line

```
PermitEmptyPAsswords no
```

- - Then restart the sshd daemon by running `systemctl restart sshd`

Limit Users' SSH Access

- To provide another layer of security, you should limit your SSH logins to only certain users who need remote access
  - Become root
  - Edit your `/etc/ssh/sshd_config` file and add the following line:
    - Go all the way down of the file by typing Shift + G in the `vi` file editor.

```
AllowUsers user1 user2
```

- Replace "user1" and "user2" with real user names.
- Any time you add any new parameters always put it a comment, in that way you know waht the next line will do or be about. For example `# Allow the following user only`, then you add the desired line.
  - Then restart the sshd daemon by running `systemctl restart sshd`

Use a different port

- By default SSH port runs on 22. Most hackers looking for any open SSH servers will look for port 22 and changing can make the system much more secure
  - Become root
  - Edit your `/etc/ssh/sshd_config` file and remove # from the following line and change the port number

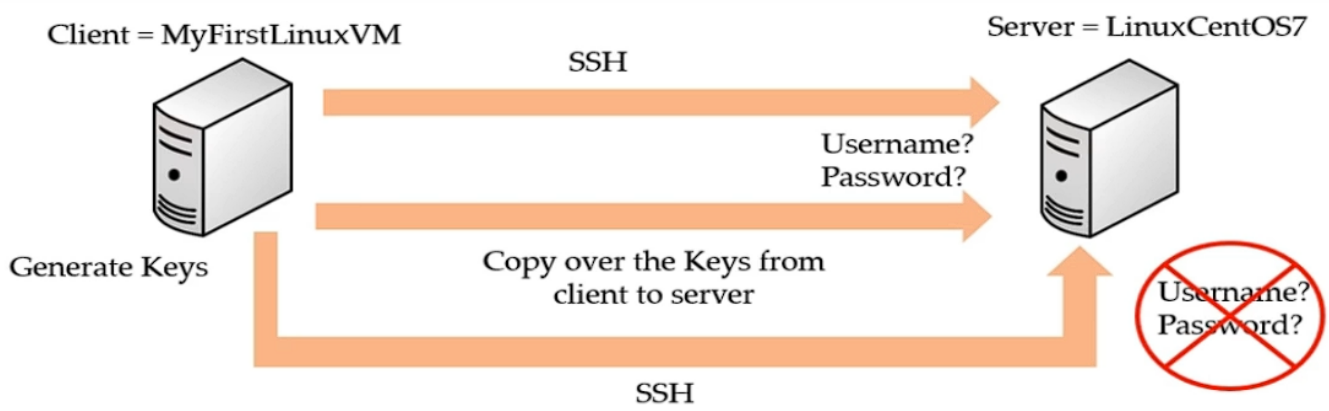
```
Port 22
```

- Change the port from 22 to any other port number.
  - Make sure whatever the port that you're assigning to this SSH is not being used by any other programs.
  - You could search online to see which ports are available.
  - Then restart the sshd daemon by running `systemctl restart sshd`
- If trying to ssh through PuTTY remember to change the port as well to the port you assigned.

SSH-Keys - Access Remote Server without Password

- If we have a Linux machine and we want to access another Linux machine. That another Linux machine is for us a remote machine.
- Two reasons to access a remote machine
  - **Repetitive logins** - logging from machine A to machine B in repetitive times, maybe 10, or 20 times a day, and you do not want to enter a username and password again and again.
  - Automation through scripts - If you have scripts that are sitting on your server A and you need to execute those scripts on server B, so that is an automation that will run on B, buy every time you run it from A to B it will prompt a username and password.

- Keys are generated at user level
  - mmarin
  - root



- You can see this process as a person who wants to enter a house but every time he wants to and does not have the key he has to ring the bell (input credentials)
- But if he goes through a verification process and actually hands the key to the house owner he will now be verified and won't need to ring the bell again.

Steps to get this process to work:

Client = MyFirstLinuxVM

- Step 1 - Generate the Key
  - Run `ssh-keygen`
  - On the client machine (the one that is trying to go to the server), we have to generate the Keys.
- Step 2 - Copy the key to the server.
  - Run `ssh-copy-id root@192.168.1.x`
    - `root` is included because you generated the keys as root so you specify as root and @ the IP address of the server that you are copying to.
- Step 3 - Login from client to server.
  - Run `ssh root@192.168.1.x`
  - Or alternatively run `ssh -l root 192.168.1.x`
    - Both commands are the same, they actually let you in from one machine to another, on the account root.

Example using `ssh-keygen` command:

```
[root@localhost ~]# ssh-keygen
```

- Command to generate SSH keys.

Output:

```
Generating public/private rsa key pair.

Enter file in which to save the key (/home/mmarin/.ssh/id_rsa): Created directory '/home/mmarin/.ssh'.
Enter passphrase (empty for no passphrase):
Enter same passphrase again:
Your identification has been saved in /home/mmarin/.ssh/id_rsa
Your public key has been saved in /home/mmarin/.ssh/id_rsa.pub
The key fingerprint is:
SHA256:AZa1/m8QkigZ0r4D/aU3Bxj8GTliqGaKG11zsVKJbJY mmarin@localhost.localdomain
The key's randomart image is:
+---[RSA 3072]----+
|   o =o+..       |
|  . E.X.+.       |
|   B = Oo=       |
|  = O =.B..      |
|.= o B oSo .     |
|+ . o o o.o      |
| o   . . o..     |
|.          ..    |
|           ..    |
+----[SHA256]-----+
```

- By default the system is going to save the key in the `/root/.ssh/id_rsa/` directory (On local machine or client)
  - If you wanted to change this location you could specify it at the moment it is prompted.
  - To leave it default just simply hit Enter.
- Next step is asking you to enter passphrase.
  - If you are in the production environment, in the corporate environment it is recommended to put in a passphrase. Anything that you like could be that passphrase.
  - To leave it empty just hit Enter.
- After those prompts the key is displayed and saved to a file that is also displayed for the user to use.
- Now we need to copy this key to our server, so next time we log into the server, the server would know and let you in without a password.

Example using `ssh-copy-id` command:

```
[root@localhost ~]# ssh-copy-id root@192.168.1.58
```

- Attempting to copy the key to the root user of the 192.168.1.58 server.
- This IP is only used for the sake of exemplification.

Output:

```
/usr/bin/ssh-copy-id: INFO: Source of key(s) to be installed: "/home/mmarin/.ssh/id_rsa.pub"
/usr/bin/ssh-copy-id: INFO: attempting to log in with the new key(s), to filter out any that are
already installed
/usr/bin/ssh-copy-id: INFO: 1 key(s) remain to be installed --if you are prompted now it is to install
the new keys
root@192.168.1.58's password:
```

- Now it is going to actually ask you for the root password of the server because you are going in for the first time. And of course once you add that key, it's gonna ask you that as just the last time.

Output after password:

```
Number of key(s) added: 1

Now try logging into the machine, with: "ssh 'root@192.168.1.58'"
and check to make sure that only the key(s) you wanted were added.
```

- Now that key is added to the server and it is added to `/root/.ssh/authorized_keys` file.
- You could actually go into that directory, `/root/.ssh/` by doing `cd /root/.ssh/` on the server and then running `ls -l` command to see the contents of the directory.
    - A file named "authorized_keys" will be in there.

Example using `cat` command:

```
[root@server ~]$ cat /root/.ssh/authorized_keys
```

- Using `cat` command to check the contents of the `/root/.ssh/authorized_keys` file (Checking what the actual key is).

Output:

```
ssh-rsa AAAAB3NzaClyc2EAAAADAQABAAABAQDtZvCgdvvlwQi18n1r0jqGkob
D5jBGnZl/J4lPoobw3Hk8Ro8b0x9wv/TQtauUpxralTUx2YI6z5nUT2e00a4/Yb
s0yBlLfdYXX8z/60kau7+BOYjq3IfV57dEWoCs37DmS5xLi5QyBsCvPYYaneWP/
ri3sphT35n8ICd84n0YZSti7RxsxBz/jR/18iDZMEeQvNcCAVG0OTSz2pX/9dzs
9GdpeSFFopYpf6eAPRxx76Tn7LSX8Tg92ws1nrAZGvX38szTgbB4yI3rq0uMqxn
pruRwptwCsfSpTnWiycq7xjwGCS2zh/D0KGiS/bxLZJEVOLtkudCblzLXqx/0v1
OX root@MyFirstLinuxVM
```

- This entire key has been copied as root from the machine "MyFirstLinuxVM" (Last line of the file)
- Now if you come and SSH from the "MyFirstLInuxVM" to the server you are not going to be prompted a password.