FR. CONCEICAO RODRIGUES COLLEGE OF ENGINEERING

Department of Computer Engineering

1. Course, Subject & Experiment Details

Practical No:	**8**
Title:	**Configuring SSH servers on Linux Machine**
Name of the Student:	**Warren Fernandes**
Roll No:	**8940**
Date of Performance:	**31-01-2022**
Date of Submission:	**11-04-2022**

Evaluation:

Sr. No.	Rubric	Grade
1	On time submission/completion (2)	
2	Preparedness (2)	
3	Skill (4)	
4	Output (2)	

Signature of the Teacher

What is SSH?

- SSH stands for Secure Shell.
- SSH is a network protocol for secure data communication.
- SSH protocol allows remote command-line login.
- SSH protocol enables remote command execution.
- To use SSH you need to deploy SSH Server and SSH Client program respectively.
- OpenSSH is a FREE version of the SSH.
- Telnet, rlogin, and FTP transmit unencrypted data over the internet.
- OpenSSH encrypts data before sending it over insecure network like the internet.
- OpenSSH effectively eliminates eavesdropping, connection hijacking, and other attacks.
- OpenSSH provides secure tunnelling and several authentication methods.
- OpenSSH replaces Telnet and rlogin with SSH, RCP with SCP, FTP with sftp.

Terminologies used

sshd: The daemon service that implements the ssh server. By default it must be listening on port 22TCP/IP.

ssh: The ssh [Secure Shell command] is a secure way to log and execute commands in to SSH Serversystem.

scp: The Secure Copy command is a secure way to transfer files between computers using the private/public key encryption method.

ssh-keygen: This utility is used to create the public/private keys.

ssh-agent: This utility holds private keys used for RSA authentication.

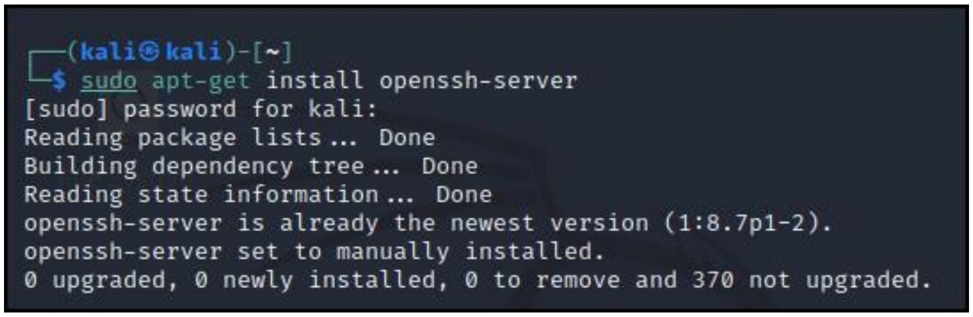
ssh-add: Adds RSA identities to the authentication agent ssh-agent.

Requirements:

- Two Linux Virtual Machines installed on Virtual-Box or VMware
- Openssh-server installed on both the VMs (if not then follow next section)
- To have both the VMs have different IP address make sure to enable NAT Networksettings in VirtualBox.

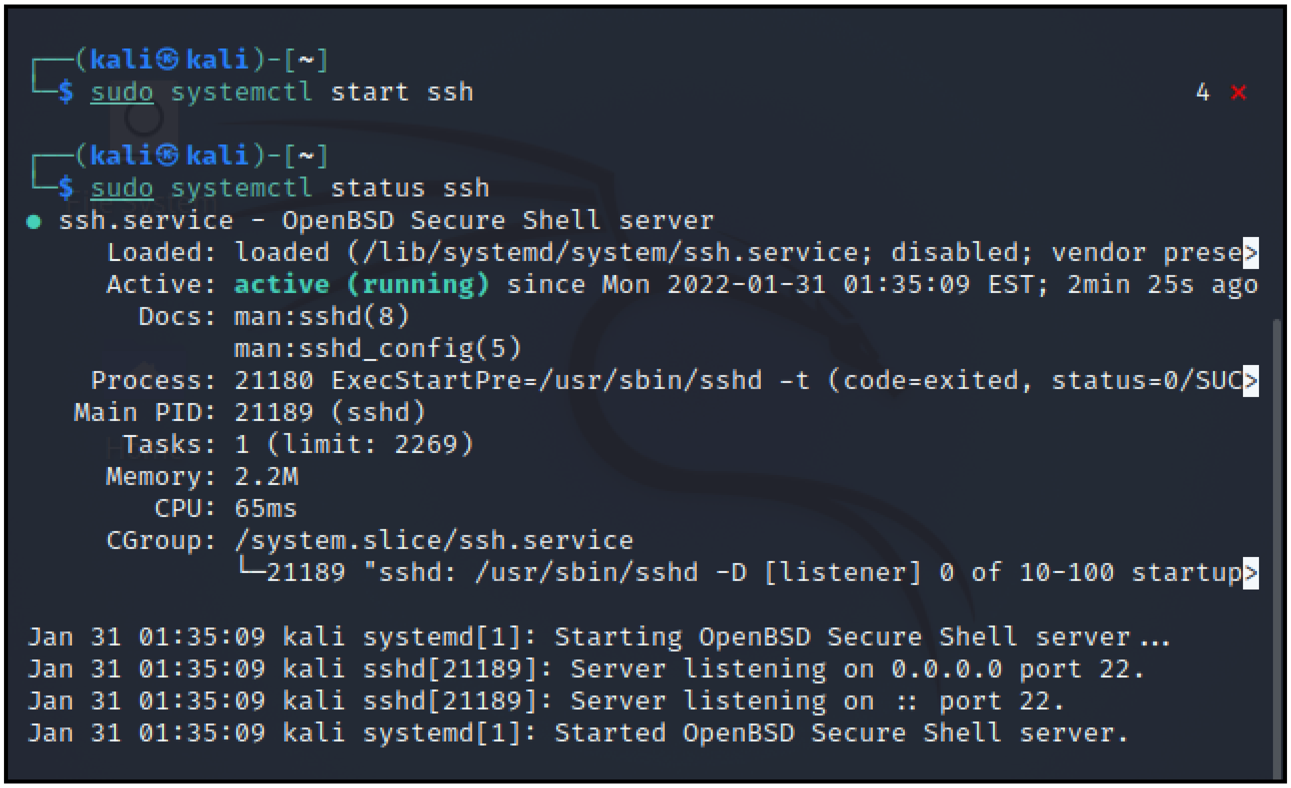
How to configure SSH client on Linux?

Install openssh-server using apt-get



```
┌──(kali㉿kali)-[~]
└─$ sudo apt-get install openssh-server
[sudo] password for kali:
Reading package lists... Done
Building dependency tree... Done
Reading state information... Done
openssh-server is already the newest version (1:8.7p1-2).
openssh-server set to manually installed.
0 upgraded, 0 newly installed, 0 to remove and 370 not upgraded.
```

Check the current status of **sshd** service, it must be running. If service is stopped start it. Options you need with service command are **start | stop | restart | status**

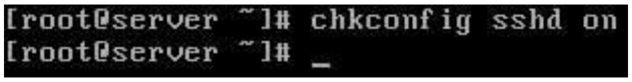


```
┌──(kali㉿kali)-[~]
└─$ sudo systemctl start ssh                                              4 ×

┌──(kali㉿kali)-[~]
└─$ sudo systemctl status ssh
● ssh.service - OpenBSD Secure Shell server
     Loaded: loaded (/lib/systemd/system/ssh.service; disabled; vendor prese>
     Active: active (running) since Mon 2022-01-31 01:35:09 EST; 2min 25s ago
       Docs: man:sshd(8)
             man:sshd_config(5)
    Process: 21180 ExecStartPre=/usr/sbin/sshd -t (code=exited, status=0/SUC>
   Main PID: 21189 (sshd)
      Tasks: 1 (limit: 2269)
     Memory: 2.2M
        CPU: 65ms
     CGroup: /system.slice/ssh.service
             └─21189 "sshd: /usr/sbin/sshd -D [listener] 0 of 10-100 startup>

Jan 31 01:35:09 kali systemd[1]: Starting OpenBSD Secure Shell server...
Jan 31 01:35:09 kali sshd[21189]: Server listening on 0.0.0.0 port 22.
Jan 31 01:35:09 kali sshd[21189]: Server listening on :: port 22.
Jan 31 01:35:09 kali systemd[1]: Started OpenBSD Secure Shell server.
```

*NOTE: if above command show error then use –

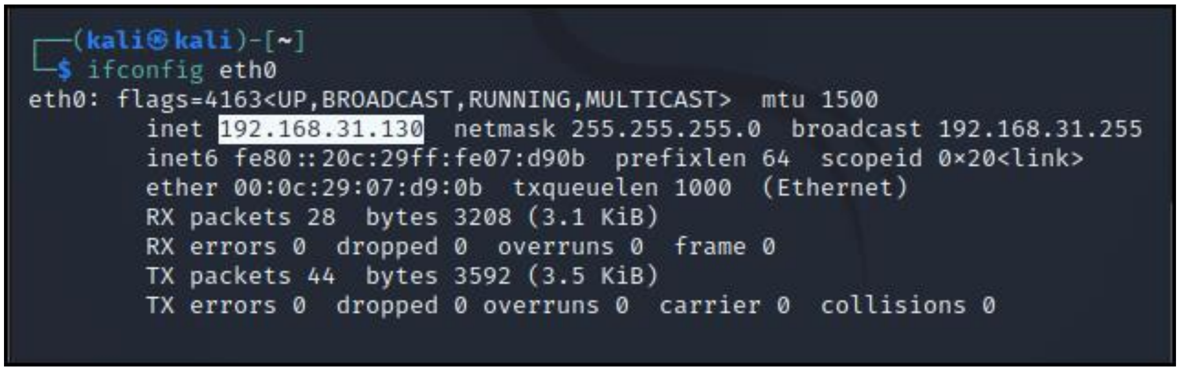
“sudo systemctl start ssh” OR “sudo systemctl status ssh” (without double quotes)

Configure it to start when the system is booted - **sudo systemctl enable ssh**



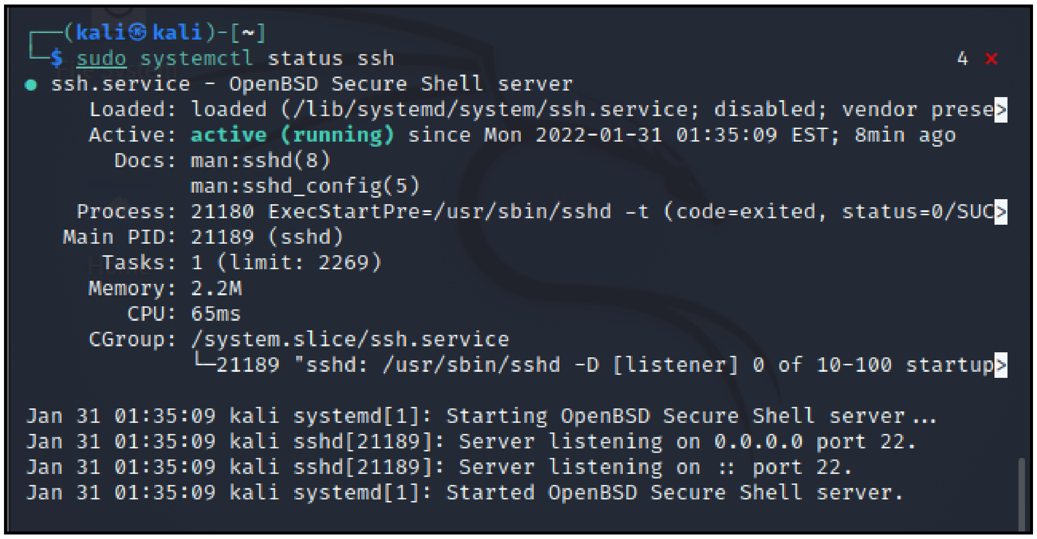
```
[root@server ~]# chkconfig sshd on
[root@server ~]# _
```

IP address of OpenSSH server is required, note it down



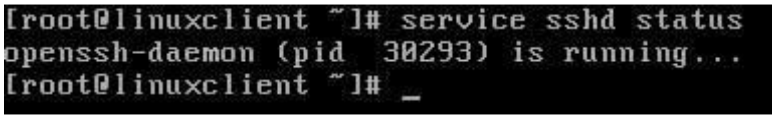
```
┌──(kali㉿kali)-[~]
└─$ ifconfig eth0
eth0: flags=4163<UP,BROADCAST,RUNNING,MULTICAST>  mtu 1500
        inet 192.168.31.130  netmask 255.255.255.0  broadcast 192.168.31.255
        inet6 fe80::20c:29ff:fe07:d90b  prefixlen 64  scopeid 0x20<link>
        ether 00:0c:29:07:d9:0b  txqueuelen 1000  (Ethernet)
        RX packets 28  bytes 3208 (3.1 KiB)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 44  bytes 3592 (3.5 KiB)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0
```

Check **sshd** service status it must be running. Start it if it is off



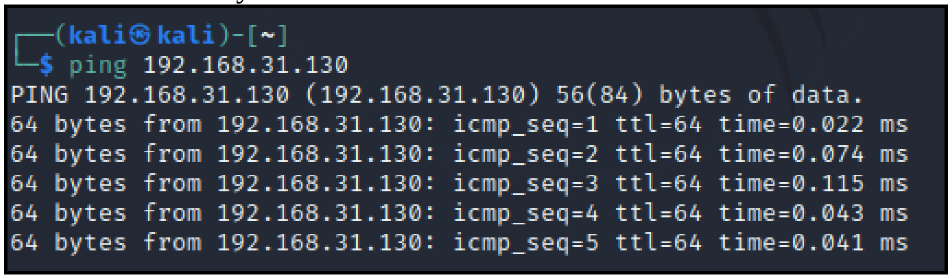
```
┌──(kali㉿kali)-[~]
└─$ sudo systemctl status ssh                                          4 ×
● ssh.service - OpenBSD Secure Shell server
     Loaded: loaded (/lib/systemd/system/ssh.service; disabled; vendor prese>
     Active: active (running) since Mon 2022-01-31 01:35:09 EST; 8min ago
       Docs: man:sshd(8)
             man:sshd_config(5)
    Process: 21180 ExecStartPre=/usr/sbin/sshd -t (code=exited, status=0/SUC>
   Main PID: 21189 (sshd)
      Tasks: 1 (limit: 2269)
     Memory: 2.2M
        CPU: 65ms
     CGroup: /system.slice/ssh.service
             └─21189 "sshd: /usr/sbin/sshd -D [listener] 0 of 10-100 startup>

Jan 31 01:35:09 kali systemd[1]: Starting OpenBSD Secure Shell server...
Jan 31 01:35:09 kali sshd[21189]: Server listening on 0.0.0.0 port 22.
Jan 31 01:35:09 kali sshd[21189]: Server listening on :: port 22.
Jan 31 01:35:09 kali systemd[1]: Started OpenBSD Secure Shell server.
```

Configure **sshd** service to start to at boot time (optional)



```
[root@linuxclient ~]# service sshd status
openssh-daemon (pid  30293) is running...
[root@linuxclient ~]# _
```

Check connectivity from SSH server

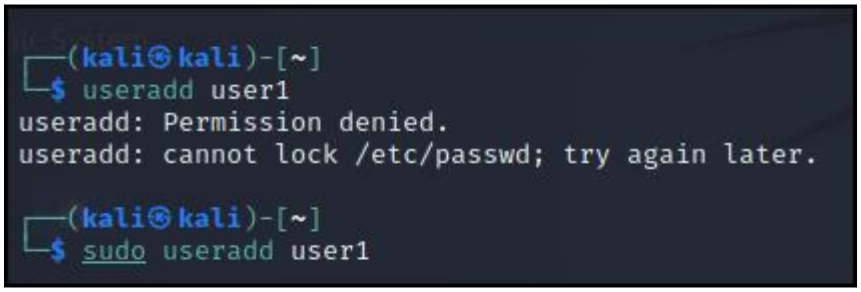


```
┌──(kali㉿kali)-[~]
└─$ ping 192.168.31.130
PING 192.168.31.130 (192.168.31.130) 56(84) bytes of data.
64 bytes from 192.168.31.130: icmp_seq=1 ttl=64 time=0.022 ms
64 bytes from 192.168.31.130: icmp_seq=2 ttl=64 time=0.074 ms
64 bytes from 192.168.31.130: icmp_seq=3 ttl=64 time=0.115 ms
64 bytes from 192.168.31.130: icmp_seq=4 ttl=64 time=0.043 ms
64 bytes from 192.168.31.130: icmp_seq=5 ttl=64 time=0.041 ms
```

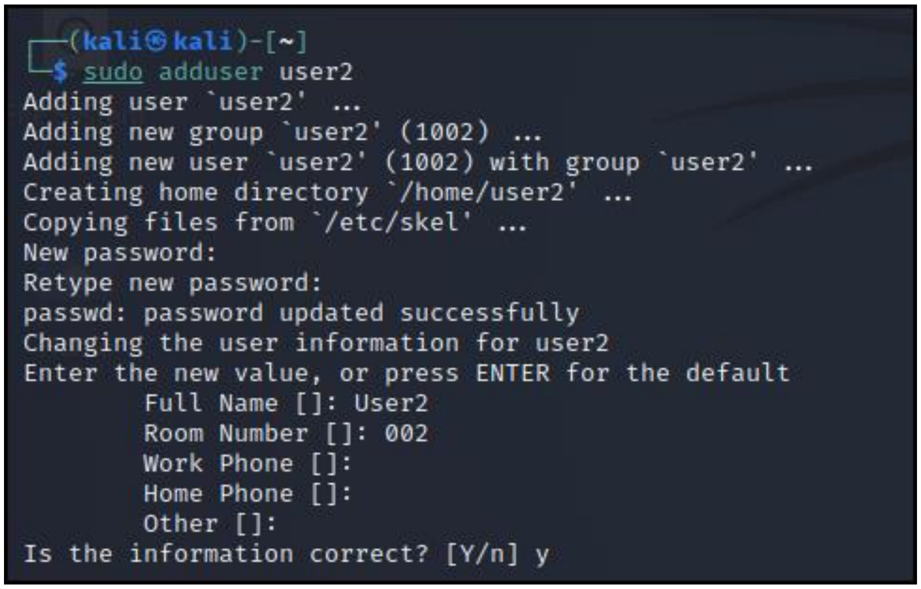
That's all setting which we need on client system.

Create two user user1 and user2 and verify that both users can login in SSH server from SSH client.

Go on server and create two users **user1** and **user2 (Use sudo -otherwise it throws error)**
Adduser – interactive->home directory will be created
Useradd – not interative ->home directory will not be created

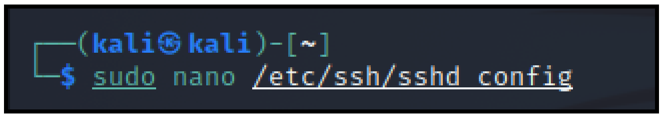


```
┌──(kali㉿kali)-[~]
└─$ useradd user1
useradd: Permission denied.
useradd: cannot lock /etc/passwd; try again later.

┌──(kali㉿kali)-[~]
└─$ sudo useradd user1
```

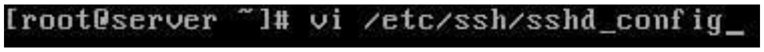


```
┌──(kali㉿kali)-[~]
└─$ sudo adduser user2
Adding user `user2' ...
Adding new group `user2' (1002) ...
Adding new user `user2' (1002) with group `user2' ...
Creating home directory `/home/user2' ...
Copying files from `/etc/skel' ...
New password:
Retype new password:
passwd: password updated successfully
Changing the user information for user2
Enter the new value, or press ENTER for the default
        Full Name []: User2
        Room Number []: 002
        Work Phone []:
        Home Phone []:
        Other []:
Is the information correct? [Y/n] y
```

Open main configuration file **sshd_config**



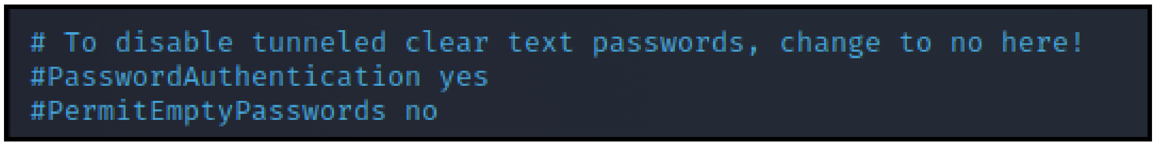
```
┌──(kali㉿kali)-[~]
└─$ sudo nano /etc/ssh/sshd_config
```



```
[root@server ~]# vi /etc/ssh/sshd_config_
```

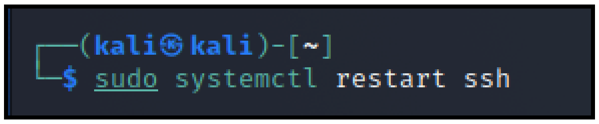
*NOTE: ‘vi’ / ‘vim’ is a CMD line editor.

Check the value of **PasswordAuthentication** directive. In order to accept local user password base authentication, it must be set to **yes**. Set it to **yes** if it is set to **no** and save the file.



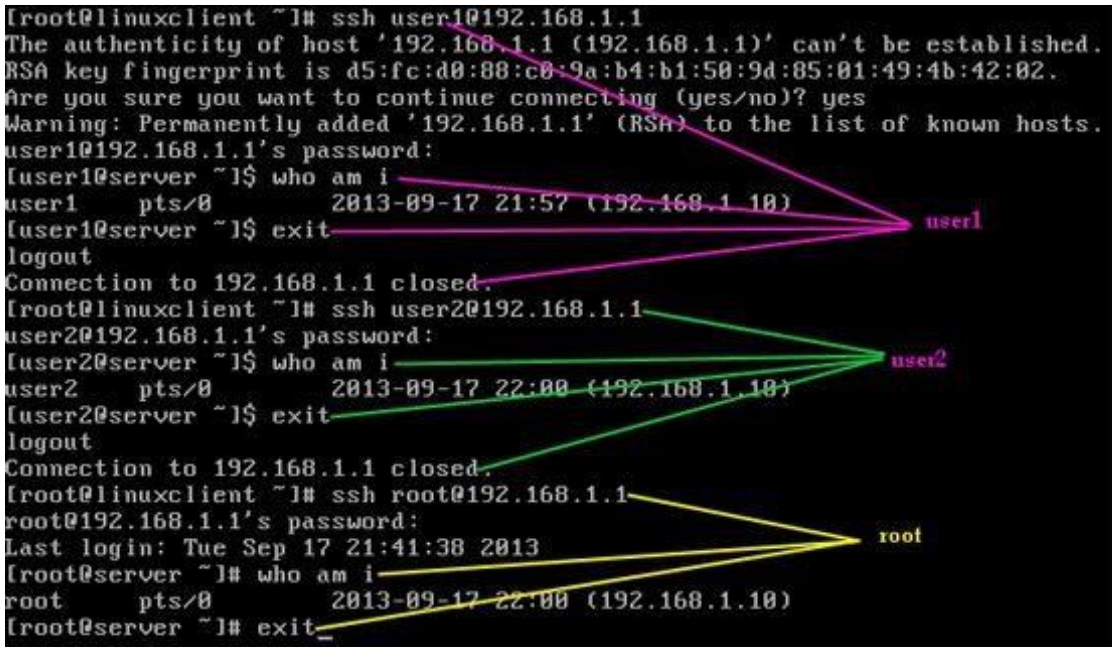
```
# To disable tunneled clear text passwords, change to no here!
#PasswordAuthentication yes
#PermitEmptyPasswords no
```

Restart the service if you have made any change in **sshd_config**



```
┌──(kali㉿kali)-[~]
└─$ sudo systemctl restart ssh
```

Go on **linuxclient** system and verify that both users can login in SSH server. Also verifyfrom **root** user.



```
[root@linuxclient ~]# ssh user1@192.168.1.1
The authenticity of host '192.168.1.1 (192.168.1.1)' can't be established.
RSA key fingerprint is d5:fc:d0:88:c0:9a:b4:b1:50:9d:85:01:49:4b:42:02.
Are you sure you want to continue connecting (yes/no)? yes
Warning: Permanently added '192.168.1.1' (RSA) to the list of known hosts.
user1@192.168.1.1's password:
[user1@server ~]$ who am i
user1    pts/0        2013-09-17 21:57 (192.168.1.10)
[user1@server ~]$ exit
logout
Connection to 192.168.1.1 closed.
[root@linuxclient ~]# ssh user2@192.168.1.1
user2@192.168.1.1's password:
[user2@server ~]$ who am i
user2    pts/0        2013-09-17 22:00 (192.168.1.10)
[user2@server ~]$ exit
logout
Connection to 192.168.1.1 closed.
[root@linuxclient ~]# ssh root@192.168.1.1
root@192.168.1.1's password:
Last login: Tue Sep 17 21:41:38 2013
[root@server ~]# who am i
root     pts/0        2013-09-17 22:00 (192.168.1.10)
[root@server ~]# exit_
```

Do not allow root and user1 users to login to it and allow the rest of users. To confirm it loginfrom user2.

User and Host Based Security

Following additional directives can be added to **/etc/sshd/sshd_config** file in order to make thessh server more restrictive.

Block empty passwords

```
PermitEmptyPasswords no
```

Block root user to log on to the system using ssh.

```
PermitRootLogin no
```

Limit the users allowed to access a system via SSH. In this case, only users 'laxmi' and 'vinita' are allowed to login on the system using SSH

```
AllowUsers laxmi vinita
```

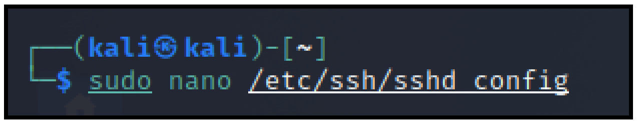
Make it more restrictive and add node address with user name. In following case only allow login through SSH users 'laxmi' and 'vinita' from 192.168.1.10 node.

```
AllowUsers laxmi@192.168.1.10 vinita@192.168.1.10
```

In addition you can restrict the access to users. In this case all users except 'user1' are allowed to connect to the SSH server.

```
DenyUsers user1
```

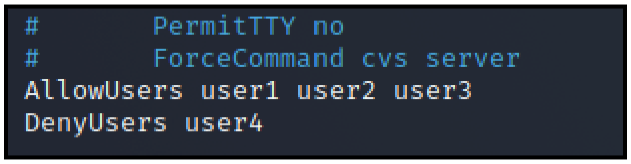
Go back on server and open main configuration file again



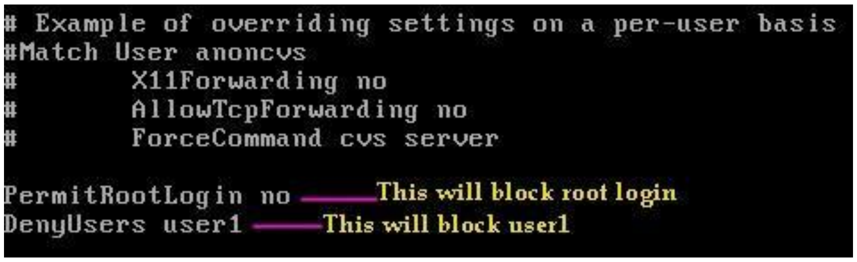
```
┌──(kali㉿kali)-[~]
└─$ sudo nano /etc/ssh/sshd_config
```

In the end of file add following directives and save the file

```
PermitRootLogin no
DenyUsers user1
```

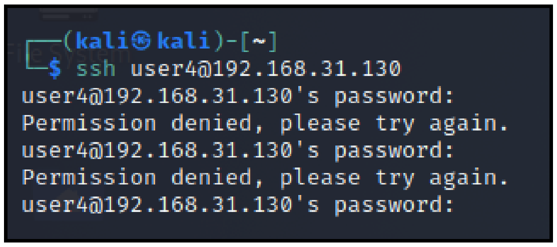


```
#       PermitTTY no
#       ForceCommand cvs server
AllowUsers user1 user2 user3
DenyUsers user4
```



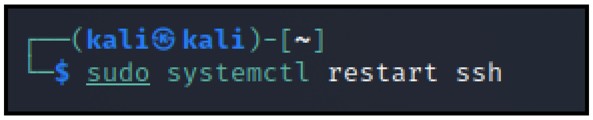
```
# Example of overriding settings on a per-user basis
#Match User anoncvs
#       X11Forwarding no
#       AllowTcpForwarding no
#       ForceCommand cvs server

PermitRootLogin no
DenyUsers user1
```

DenyUsers user4



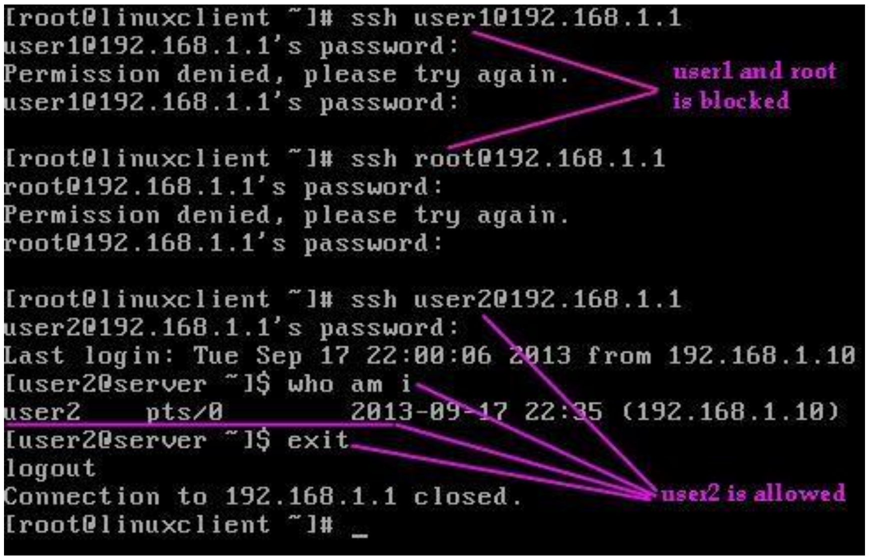
```
┌──(kali㉿kali)-[~]
└─$ ssh user4@192.168.31.130
user4@192.168.31.130's password:
Permission denied, please try again.
user4@192.168.31.130's password:
Permission denied, please try again.
user4@192.168.31.130's password:
```

Restart the **sshd** service



```
┌──(kali㉿kali)-[~]
└─$ sudo systemctl restart ssh
```

Go back to the **Linux client** system and verify that we have blocked **user1** and **root**. Also verifythat **user2** able to login in SSH server.



```
[root@linuxclient ~]# ssh user1@192.168.1.1
user1@192.168.1.1's password:
Permission denied, please try again.
user1@192.168.1.1's password:

[root@linuxclient ~]# ssh root@192.168.1.1
root@192.168.1.1's password:
Permission denied, please try again.
root@192.168.1.1's password:

[root@linuxclient ~]# ssh user2@192.168.1.1
user2@192.168.1.1's password:
Last login: Tue Sep 17 22:00:06 2013 from 192.168.1.10
[user2@server ~]$ who am i
user2     pts/0        2013-09-17 22:35 (192.168.1.10)
[user2@server ~]$ exit
logout
Connection to 192.168.1.1 closed.
[root@linuxclient ~]# _
```

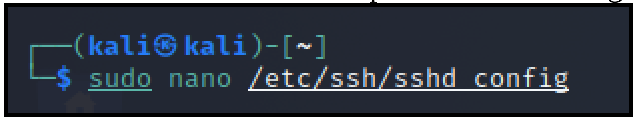
Re-configure SSH Server to allow login only using public / private keys. Generate keys for user2and verify that user2 can login using keys.

To make Linux servers more secure Linux administrators usually disable password authentication onthe SSH server and allow only public/private keys authentication.

Private Keys : Private keys are stored on the server and must be secured. Anything encrypted with the public key can only be decrypted with the paired private key. So it must be accessible only to the user owner of thatkey, in the **.ssh** subdirectory of that user's home directory.

Public Keys: Public keys are publicly available. Public keys are required to connect with the server. The public keysfor SSH servers belong to administrative workstations.

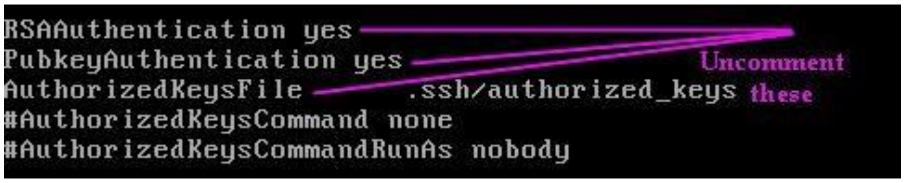
Go back on the **server** and open the main configuration file again



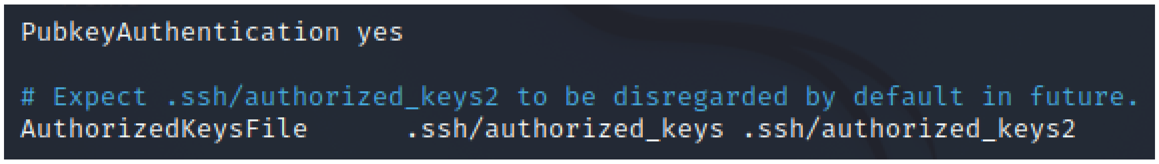
```
┌──(kali㉿kali)-[~]
└─$ sudo nano /etc/ssh/sshd_config
```

Uncomment following directives and save the file

```
RSAAuthentication yes
PubkeyAuthentication yes
AuthorizedKeysFile .ssh/authorized_keys
```

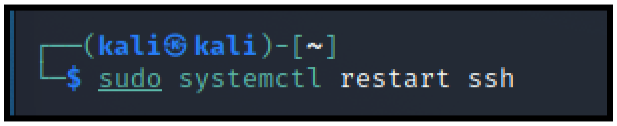


```
RSAAuthentication yes
PubkeyAuthentication yes
AuthorizedKeysFile      .ssh/authorized_keys
#AuthorizedKeysCommand none
#AuthorizedKeysCommandRunAs nobody
```



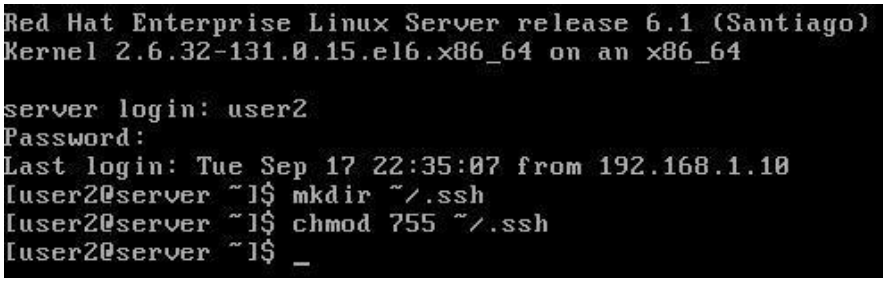
```
PubkeyAuthentication yes

# Expect .ssh/authorized_keys2 to be disregarded by default in future.
AuthorizedKeysFile      .ssh/authorized_keys .ssh/authorized_keys2
```

Restart the **sshd** service

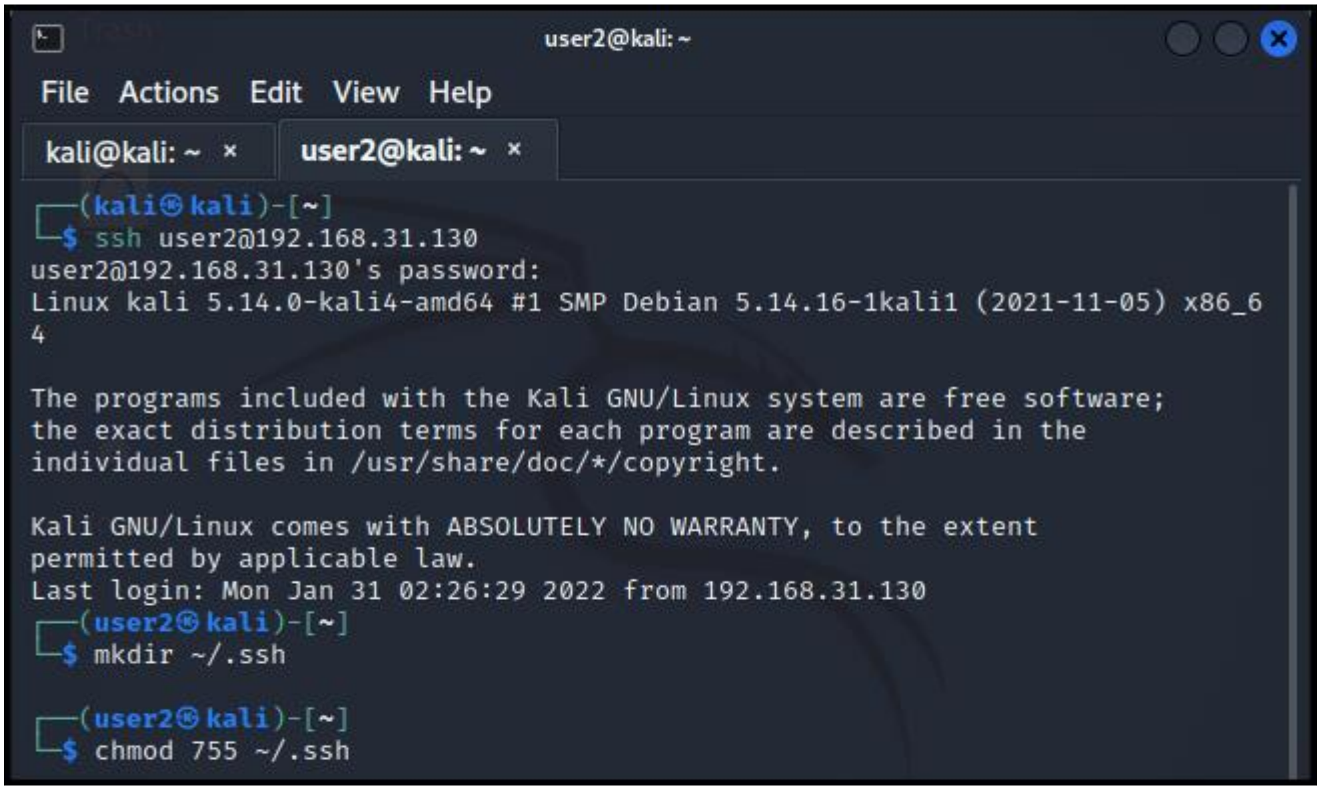


```
┌──(kali㉿kali)-[~]
└─$ sudo systemctl restart ssh
```

Login from **user2** and create an **ssh** directory with permission 755

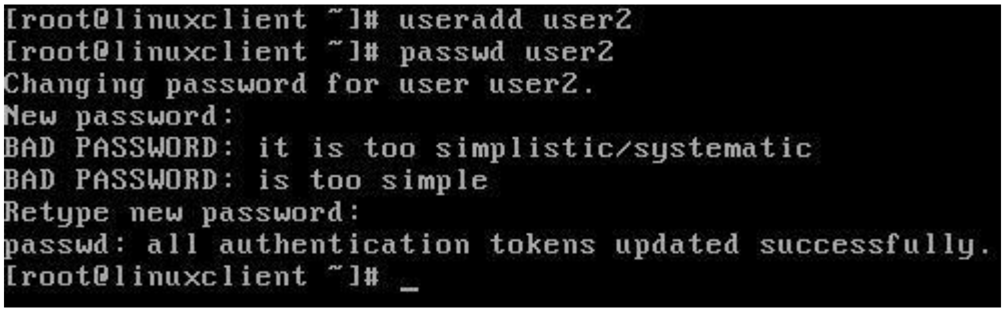


```
Red Hat Enterprise Linux Server release 6.1 (Santiago)
Kernel 2.6.32-131.0.15.el6.x86_64 on an x86_64

server login: user2
Password:
Last login: Tue Sep 17 22:35:07 from 192.168.1.10
[user2@server ~]$ mkdir ~/.ssh
[user2@server ~]$ chmod 755 ~/.ssh
[user2@server ~]$ _
```

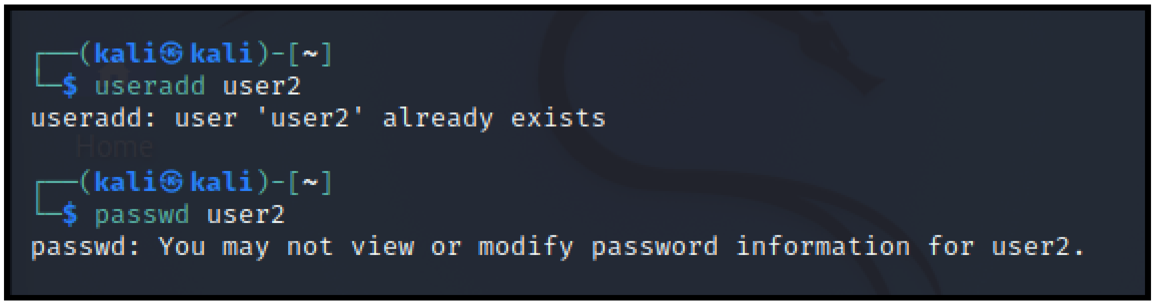


```
┌──(kali㉿kali)-[~]
└─$ ssh user2@192.168.31.130
user2@192.168.31.130's password:
Linux kali 5.14.0-kali4-amd64 #1 SMP Debian 5.14.16-1kali1 (2021-11-05) x86_6
4

The programs included with the Kali GNU/Linux system are free software;
the exact distribution terms for each program are described in the
individual files in /usr/share/doc/*/copyright.

Kali GNU/Linux comes with ABSOLUTELY NO WARRANTY, to the extent
permitted by applicable law.
Last login: Mon Jan 31 02:26:29 2022 from 192.168.31.130
┌──(user2㉿kali)-[~]
└─$ mkdir ~/.ssh

┌──(user2㉿kali)-[~]
└─$ chmod 755 ~/.ssh
```

Come back on the **linuxclient** system and create a normal user account **user2**.

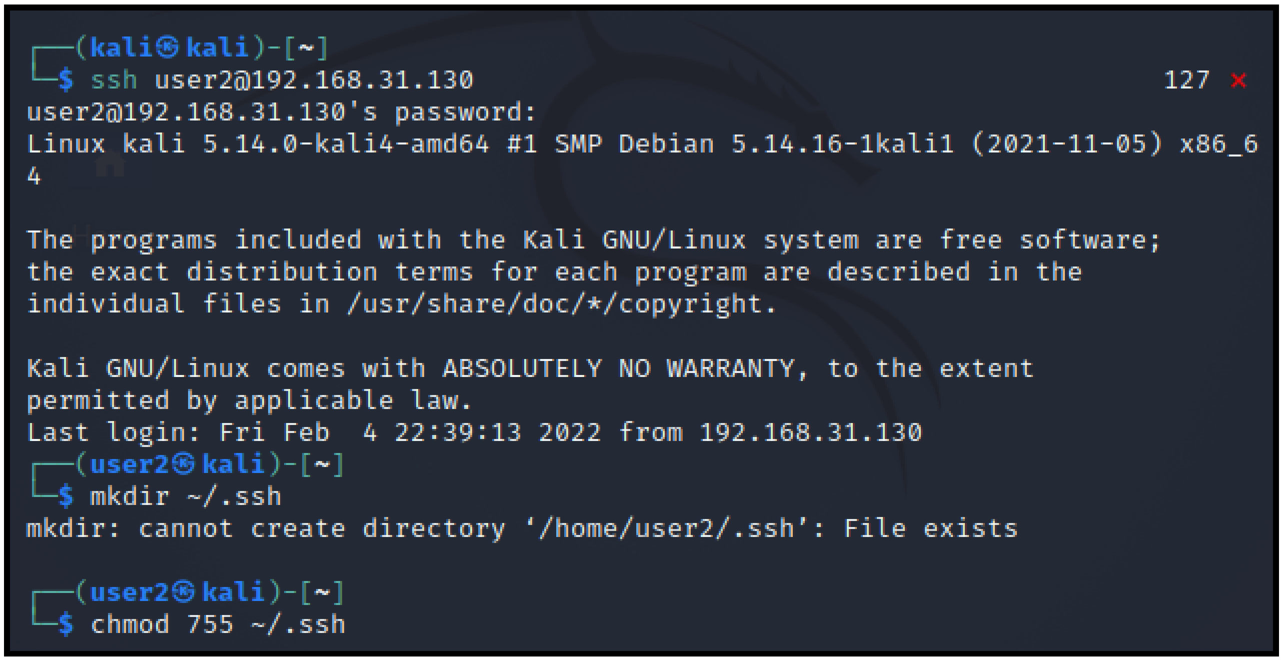


```
[root@linuxclient ~]# useradd user2
[root@linuxclient ~]# passwd user2
Changing password for user user2.
New password:
BAD PASSWORD: it is too simplistic/systematic
BAD PASSWORD: is too simple
Retype new password:
passwd: all authentication tokens updated successfully.
[root@linuxclient ~]# _
```



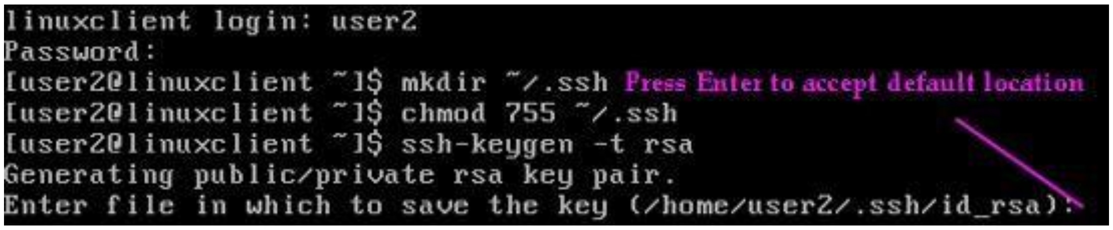
```
┌──(kali㉿kali)-[~]
└─$ useradd user2
useradd: user 'user2' already exists

┌──(kali㉿kali)-[~]
└─$ passwd user2
passwd: You may not view or modify password information for user2.
```

Login from **user2** and create an **ssh** directory with permission 755



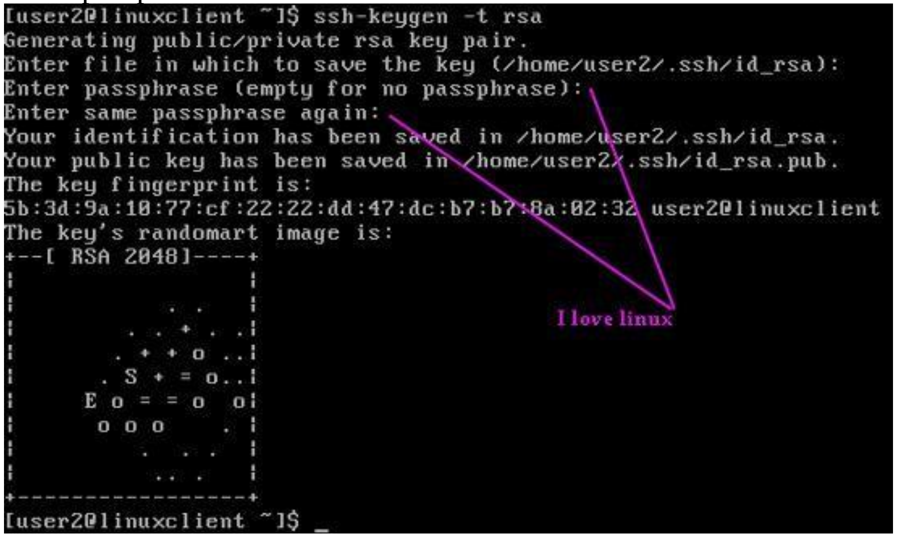
```
┌──(kali㉿kali)-[~]
└─$ ssh user2@192.168.31.130                                              127 ✗
user2@192.168.31.130's password:
Linux kali 5.14.0-kali4-amd64 #1 SMP Debian 5.14.16-1kali1 (2021-11-05) x86_6
4

The programs included with the Kali GNU/Linux system are free software;
the exact distribution terms for each program are described in the
individual files in /usr/share/doc/*/copyright.

Kali GNU/Linux comes with ABSOLUTELY NO WARRANTY, to the extent
permitted by applicable law.
Last login: Fri Feb  4 22:39:13 2022 from 192.168.31.130
┌──(user2㉿kali)-[~]
└─$ mkdir ~/.ssh
mkdir: cannot create directory '/home/user2/.ssh': File exists

┌──(user2㉿kali)-[~]
└─$ chmod 755 ~/.ssh
```

Generate the public/private key pair. Accept the default location for the key file.

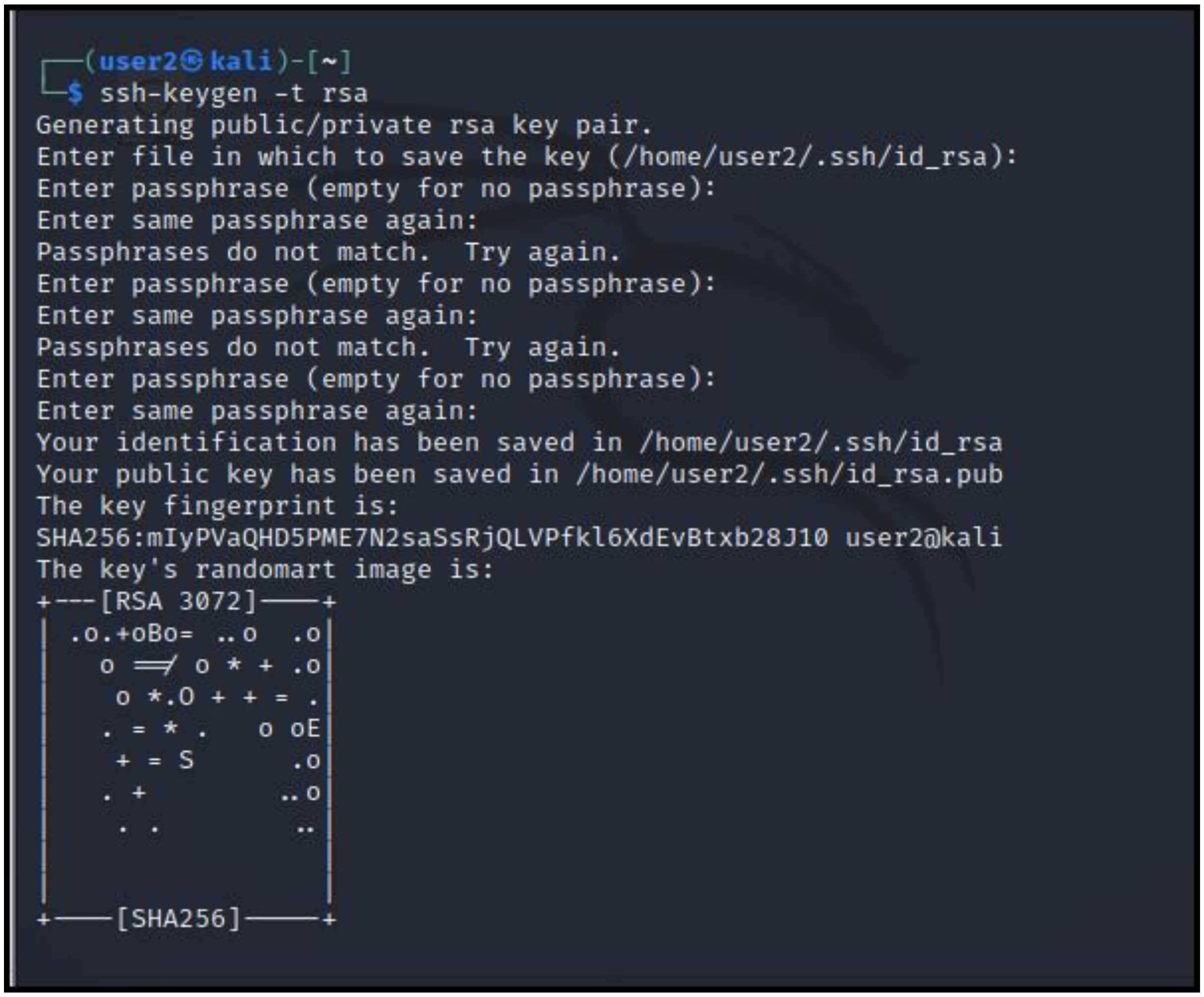


```
linuxclient login: user2
Password:
[user2@linuxclient ~]$ mkdir ~/.ssh
[user2@linuxclient ~]$ chmod 755 ~/.ssh
[user2@linuxclient ~]$ ssh-keygen -t rsa
Generating public/private rsa key pair.
Enter file in which to save the key (/home/user2/.ssh/id_rsa):
```

Enter passphrase 'I love Linux' and confirm

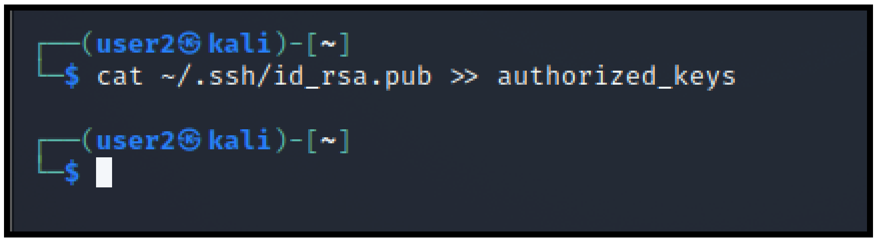


```
[user2@linuxclient ~]$ ssh-keygen -t rsa
Generating public/private rsa key pair.
Enter file in which to save the key (/home/user2/.ssh/id_rsa):
Enter passphrase (empty for no passphrase):
Enter same passphrase again:
Your identification has been saved in /home/user2/.ssh/id_rsa.
Your public key has been saved in /home/user2/.ssh/id_rsa.pub.
The key fingerprint is:
5b:3d:9a:10:77:cf:22:22:dd:47:dc:b7:b7:8a:02:32 user2@linuxclient
The key's randomart image is:
+--[ RSA 2048]----+
|                 |
|          . .    |
|       . . + . . |
|      . + + o . .|
|     . S + = o . |
|    E o = = o  o|
|     o o o    . |
|         .  . .  |
|          . . .  |
+-----------------+
[user2@linuxclient ~]$ _
```



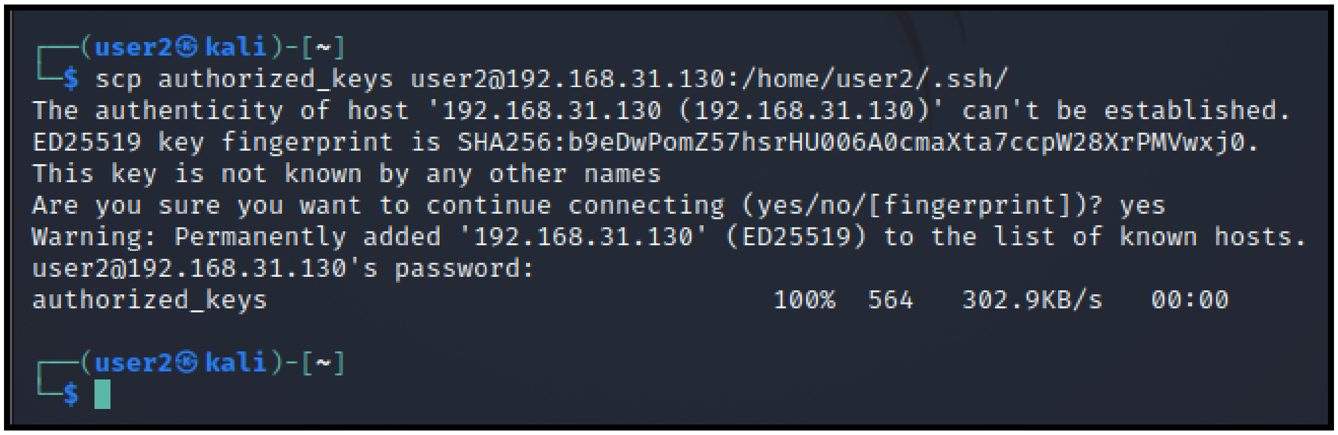
```
┌──(user2㉿kali)-[~]
└─$ ssh-keygen -t rsa
Generating public/private rsa key pair.
Enter file in which to save the key (/home/user2/.ssh/id_rsa):
Enter passphrase (empty for no passphrase):
Enter same passphrase again:
Passphrases do not match.  Try again.
Enter passphrase (empty for no passphrase):
Enter same passphrase again:
Passphrases do not match.  Try again.
Enter passphrase (empty for no passphrase):
Enter same passphrase again:
Your identification has been saved in /home/user2/.ssh/id_rsa
Your public key has been saved in /home/user2/.ssh/id_rsa.pub
The key fingerprint is:
SHA256:mIyPVaQHD5PME7N2saSsRjQLVPfkl6XdEvBtxb28J10 user2@kali
The key's randomart image is:
+---[RSA 3072]----+
| .o.+oBo= ..o  .o|
|   o ==/ o * + .o|
|    o *.O + + = .|
|   . = * .   o oE|
|    + = S      .o|
|   . +        ..o|
|    . .        ..|
|                 |
|                 |
+----[SHA256]-----+
```

The public key is stored in **/home/user2/.ssh/id_rsa.pub**. Create a copy of the public key



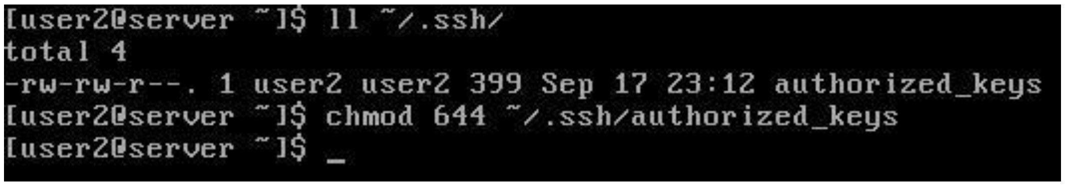
```
┌──(user2㉿kali)-[~]
└─$ cat ~/.ssh/id_rsa.pub >> authorized_keys

┌──(user2㉿kali)-[~]
└─$ 
```

Copy the **authorized_keys** file on server to **/home/user2/.ssh/authorized_keys**. Enter **user2** [user account on server] password when asked

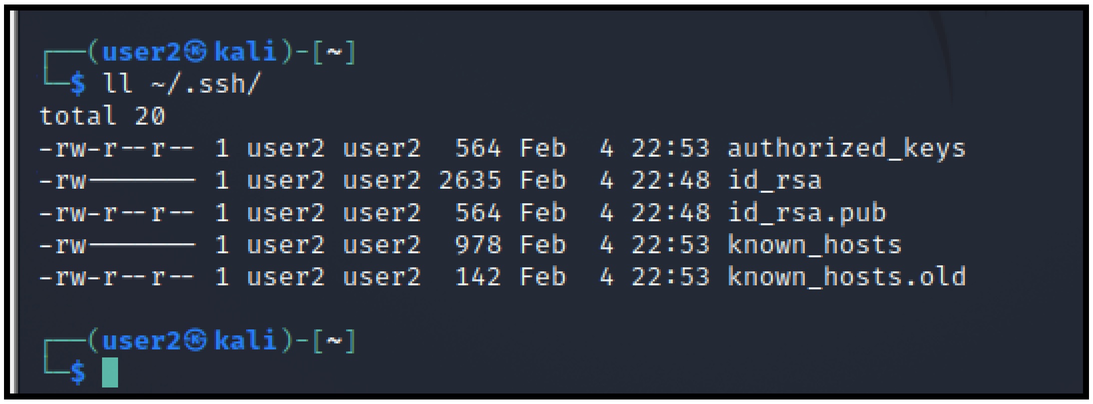


```
┌──(user2㉿kali)-[~]
└─$ scp authorized_keys user2@192.168.31.130:/home/user2/.ssh/
The authenticity of host '192.168.31.130 (192.168.31.130)' can't be established.
ED25519 key fingerprint is SHA256:b9eDwPomZ57hsrHU006A0cmaXta7ccpW28XrPMVwxj0.
This key is not known by any other names
Are you sure you want to continue connecting (yes/no/[fingerprint])? yes
Warning: Permanently added '192.168.31.130' (ED25519) to the list of known hosts.
user2@192.168.31.130's password:
authorized_keys                                 100%  564    302.9KB/s   00:00

┌──(user2㉿kali)-[~]
└─$ 
```

On **server** verify that we have successfully copied public key on the server. Also, set permission to 644for **authorized_keys**

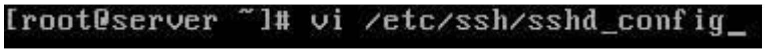


```
[user2@server ~]$ ll ~/.ssh/
total 4
-rw-rw-r--. 1 user2 user2 399 Sep 17 23:12 authorized_keys
[user2@server ~]$ chmod 644 ~/.ssh/authorized_keys
[user2@server ~]$ _
```



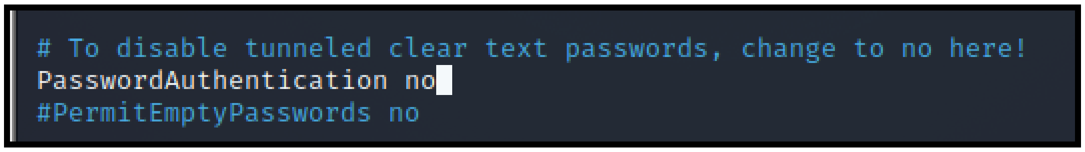
```
┌──(user2㉿kali)-[~]
└─$ ll ~/.ssh/
total 20
-rw-r--r-- 1 user2 user2  564 Feb  4 22:53 authorized_keys
-rw------- 1 user2 user2 2635 Feb  4 22:48 id_rsa
-rw-r--r-- 1 user2 user2  564 Feb  4 22:48 id_rsa.pub
-rw------- 1 user2 user2  978 Feb  4 22:53 known_hosts
-rw-r--r-- 1 user2 user2  142 Feb  4 22:53 known_hosts.old

┌──(user2㉿kali)-[~]
└─$ 
```

Login from the **root** on server and open **sshd_config** file



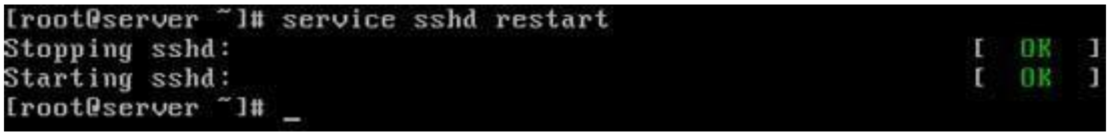
```
[root@server ~]# vi /etc/ssh/sshd_config_
```

Set **PasswordAuthentication** directive to **no** and save the file. This will block login using passwords.



```
# To disable tunneled clear text passwords, change to no here!
PasswordAuthentication no
#PermitEmptyPasswords no
```

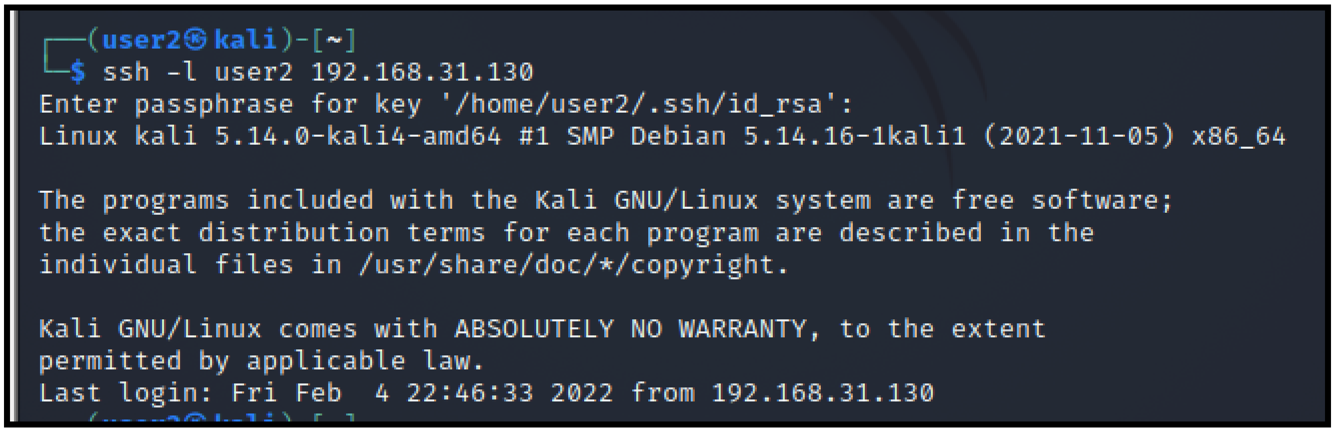
Restart the **sshd** service



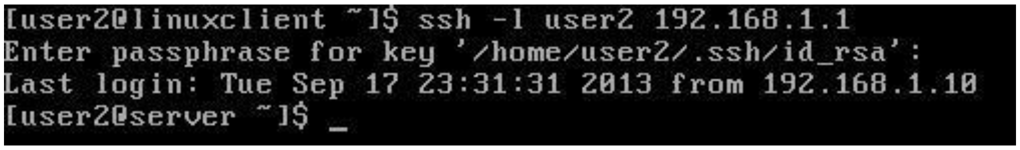
```
[root@server ~]# service sshd restart
Stopping sshd:                                             [  OK  ]
Starting sshd:                                             [  OK  ]
[root@server ~]# _
```

Come back on the **Linux client** system. Logout from **user2** and login back in.

Now try to login from **user2** on **linuxclient**. Enter passphrase 'I love linux'

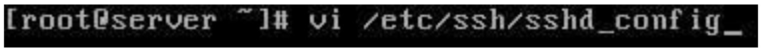


```
┌──(user2㉿kali)-[~]
└─$ ssh -l user2 192.168.31.130
Enter passphrase for key '/home/user2/.ssh/id_rsa':
Linux kali 5.14.0-kali4-amd64 #1 SMP Debian 5.14.16-1kali1 (2021-11-05) x86_64

The programs included with the Kali GNU/Linux system are free software;
the exact distribution terms for each program are described in the
individual files in /usr/share/doc/*/copyright.

Kali GNU/Linux comes with ABSOLUTELY NO WARRANTY, to the extent
permitted by applicable law.
Last login: Fri Feb  4 22:46:33 2022 from 192.168.31.130
```



```
[user2@linuxclient ~]$ ssh -l user2 192.168.1.1
Enter passphrase for key '/home/user2/.ssh/id_rsa':
Last login: Tue Sep 17 23:31:31 2013 from 192.168.1.10
[user2@server ~]$ _
```

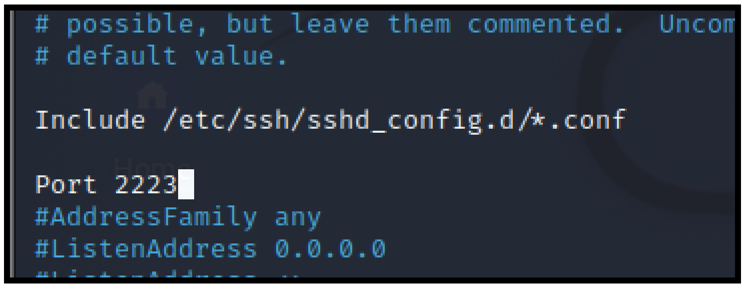
Change default ssh port to 2223
Come on the server and open the **sshd_config** file again



```
[root@server ~]# vi /etc/ssh/sshd_config_
```

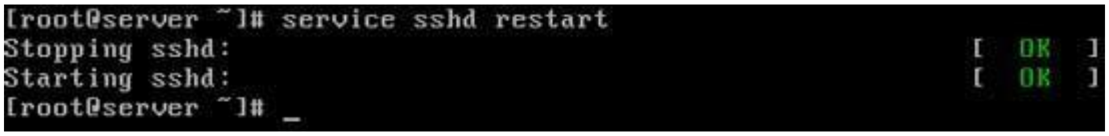
Uncomment following directive and change value to **2223**

```
#port 22
```



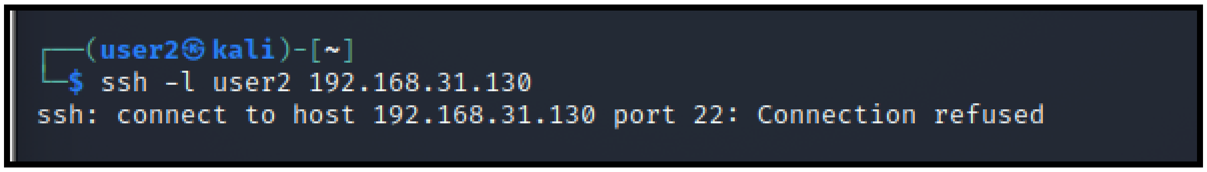
```
# possible, but leave them commented.  Uncom
# default value.

Include /etc/ssh/sshd_config.d/*.conf

Port 2223
#AddressFamily any
#ListenAddress 0.0.0.0
```

restart the **sshd** service



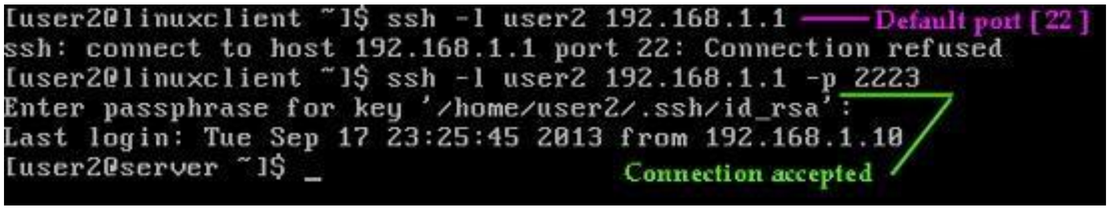
```
[root@server ~]# service sshd restart
Stopping sshd:                                             [  OK  ]
Starting sshd:                                             [  OK  ]
[root@server ~]# _
```

Go back on the **Linux client** system and try to connect with the default port

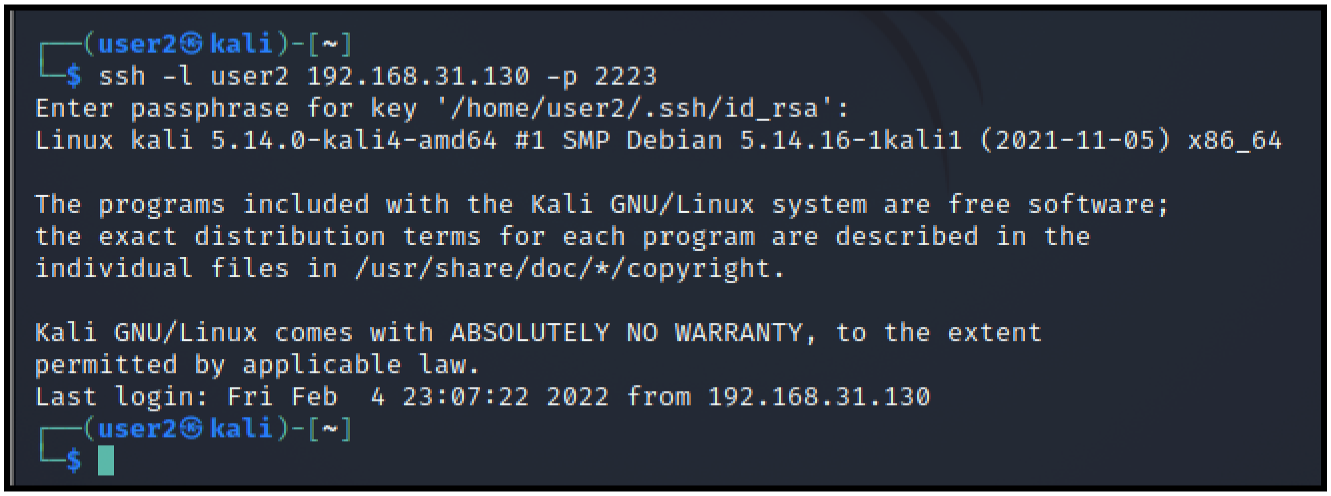


```
┌──(user2㉿kali)-[~]
└─$ ssh -l user2 192.168.31.130
ssh: connect to host 192.168.31.130 port 22: Connection refused
```

Now specify the new port



```
[user2@linuxclient ~]$ ssh -l user2 192.168.1.1
ssh: connect to host 192.168.1.1 port 22: Connection refused
[user2@linuxclient ~]$ ssh -l user2 192.168.1.1 -p 2223
Enter passphrase for key '/home/user2/.ssh/id_rsa':
Last login: Tue Sep 17 23:25:45 2013 from 192.168.1.10
[user2@server ~]$ _
```



```
┌──(user2㉿kali)-[~]
└─$ ssh -l user2 192.168.31.130 -p 2223
Enter passphrase for key '/home/user2/.ssh/id_rsa':
Linux kali 5.14.0-kali4-amd64 #1 SMP Debian 5.14.16-1kali1 (2021-11-05) x86_64

The programs included with the Kali GNU/Linux system are free software;
the exact distribution terms for each program are described in the
individual files in /usr/share/doc/*/copyright.

Kali GNU/Linux comes with ABSOLUTELY NO WARRANTY, to the extent
permitted by applicable law.
Last login: Fri Feb  4 23:07:22 2022 from 192.168.31.130
┌──(user2㉿kali)-[~]
└─$ 
```

Post lab Questions:

1. **What is the difference between SSH & Telnet?**

Ans:

Telnet	**SSH**
Telnet is the standard TCP/IP protocol for virtual terminal service. It enables you to establish a connection to a remote system in such a manner that it appears as a local system.	SSH or Secure Shell is a program to log into another computer over a network to execute commands in a remote machine.
Telnet uses port 23, which was designed specifically for local area networks	SSH runs on port 22 by default, which you can change it.
No privileges are provided for the user's authentication.	SSH is a more secure protocol, so it uses public-key encryption for authentication.
Suitable for private networks	Suitable for public networks
Telnet transfers the data in plain text.	The encrypted format should be used to send data and also uses a secure channel.
Telnet is vulnerable to security attacks.	SSH helps you to overcome many security issues of Telnet.
Required low bandwidth usage.	Required high bandwidth usage.
Data sent using this protocol cannot be easily interpreted by the hackers.	Usernames and Passwords can be prone to malicious attacks.
Used in Linux and Windows Operating system.	All popular Operating systems.

2. What is SSH port forwarding?

Ans:

One of the most useful features of SSH is port forwarding. In essence, port forwarding provides the ability to convert any insecure TCP connection into a secure SSH connection. This is also referred to as SSH tunneling. We need to know what a port is in this context. A port is an identifier of a user of TCP. So, any application that runs on top of TCP has a port number. Incoming TCP traffic is delivered to the appropriate application on the basis of the port number. An application may employ multiple port numbers. For example, for the Simple Mail Transfer Protocol (smtp), the server side generally listens on port 25, so an incoming SMTP request uses TCP and addresses the data to destination port 25. TCP recognizes that this is the SMTP server address and routes the data to the SMTP server application.

Figure below illustrates the basic concept behind port forwarding. We have a client application that is identified by port number x and a server application identified by port number y. At some point, the client application invokes the local TCP entity and requests a connection to the remote server on port y. The local TCP entity negotiates a TCP connection with the remote TCP entity, such that the connection links local port x to remote port y.

To secure this connection, SSH is configured so that the SSH Transport Layer Protocol establishes a TCP connection between the SSH client and server entities, with TCP port numbers a and b, respectively. A secure SSH tunnel is established over this TCP connection. Traffic from the client at port x is redirected to the local SSH entity and travels through the tunnel where the remote SSH entity delivers the data to the server application on port y. Traffic in the other direction is similarly redirected.

SSH supports two types of port forwarding: local forwarding and remote forwarding. Local forwarding allows the client to set up a “hijacker” process. This will intercept selected application-level traffic and redirect it from an unsecured TCP connection to a secure SSH tunnel. SSH is configured to listen on selected ports. SSH grabs all traffic using a selected port and sends it through an SSH tunnel. On the other end, the SSH server sends the incoming traffic to the destination port dictated by the client application

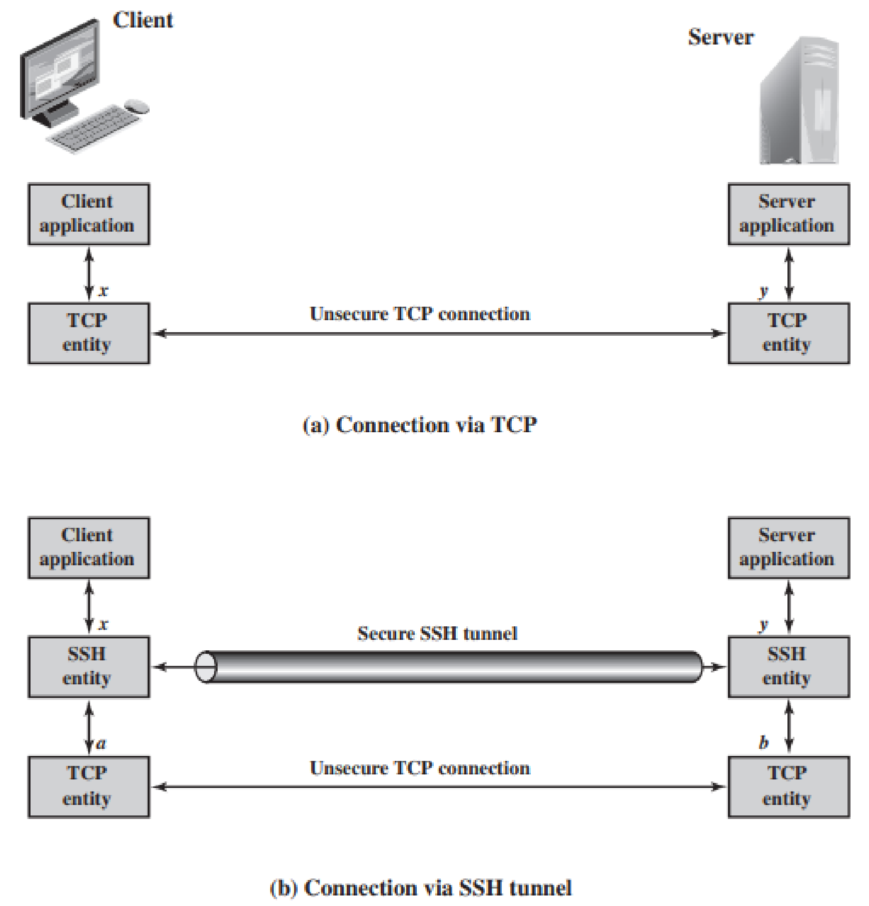


Figure SSH Transport Layer Packet Exchanges

The following example describes **local forwarding**. Suppose you have an e-mail client on your desktop and use it to get an e-mail from your mail server via the Post Office Protocol (POP). The assigned port number for POP3 is port 110. We can secure this traffic in the following way:

i. The SSH client sets up a connection to the remote server.
ii. Select an unused local port number, say 9999, and configure SSH to accept traffic from this port destined for port 110 on the server.
iii. The SSH client informs the SSH server to create a connection to the destination, in this case, mail server port 110.
iv. The client takes any bits sent to local port 9999 and sends them to the server inside the encrypted SSH session. The SSH server decrypts the incoming bits and sends the plaintext to port 110.
v. In the other direction, the SSH server takes any bits received on port 110 and sends them inside the SSH session back to the client, who decrypts and sends them to the process connected to port 9999.

With **remote forwarding**, the user's SSH client acts on the server's behalf. The client receives traffic with a given destination port number, places the traffic on the correct port and sends it to the destination the user chooses. A typical example of remote forwarding is the following. You wish to access a server at work from your home computer. Because the work server is behind a firewall, it will not accept an SSH request from your home computer. However, from work, you can set up an SSH tunnel using remote forwarding. This involves the following steps.

i. From the work computer, set up an SSH connection to your home computer. The firewall will allow this because it is a protected outgoing connection.
ii. Configure the SSH server to listen on a local port, say 22, and to deliver data across the SSH connection addressed to the remote port, say 2222.
iii. You can now go to your home computer, and configure SSH to accept traffic on port 2222.
iv. You now have an SSH tunnel that can be used for remote login to the work server.

3. How to enable password less SSH authentication in Linux?
Ans:

SSH (Secure Shell) allows secure remote connections between two systems. With this cryptographic protocol, you can manage machines, copy, or move files on a remote server via encrypted channels.

There are two ways to login onto a remote system over SSH – using **password authentication** or **public key authentication** (passwordless SSH login).

In this tutorial, you will find out how to set up and enable passwordless SSH login.

Prerequisites

- Access to command line/terminal window
- User with **sudo** or **root** privileges
- A local server and a remote server
- **SSH access** to a remote server via command line/terminal window

Before You Start: Check for Existing SSH Keys

You may already have an SSH key pair generated on your machine. To see whether you have SSH keys on the system, run the command:

```
ls -al ~/.ssh/id_*.pub
```

If the output tells you there are no such files, move on to the next step, which shows you how to generate SSH keys.

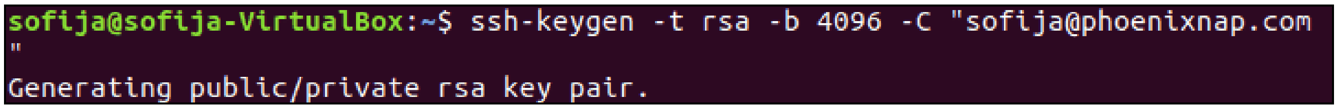
In case you do have them, you can use the existing keys, back them up and create a new pair or overwrite it.

Step 1: Generate SSH Key Pair

1. The first thing you need to do is **generate an SSH key pair** on the machine you are currently working on.

In this example, we generate a 4096-bit key pair. We also add an email address, however this is optional. The command is:

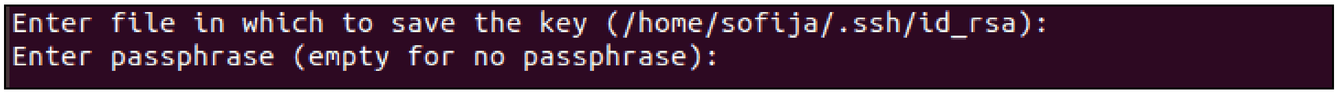
```
ssh-keygen -t rsa -b 4096 -C "your_email@domain.com"
```



```
sofija@sofija-VirtualBox:~$ ssh-keygen -t rsa -b 4096 -C "sofija@phoenixnap.com
"
Generating public/private rsa key pair.
```

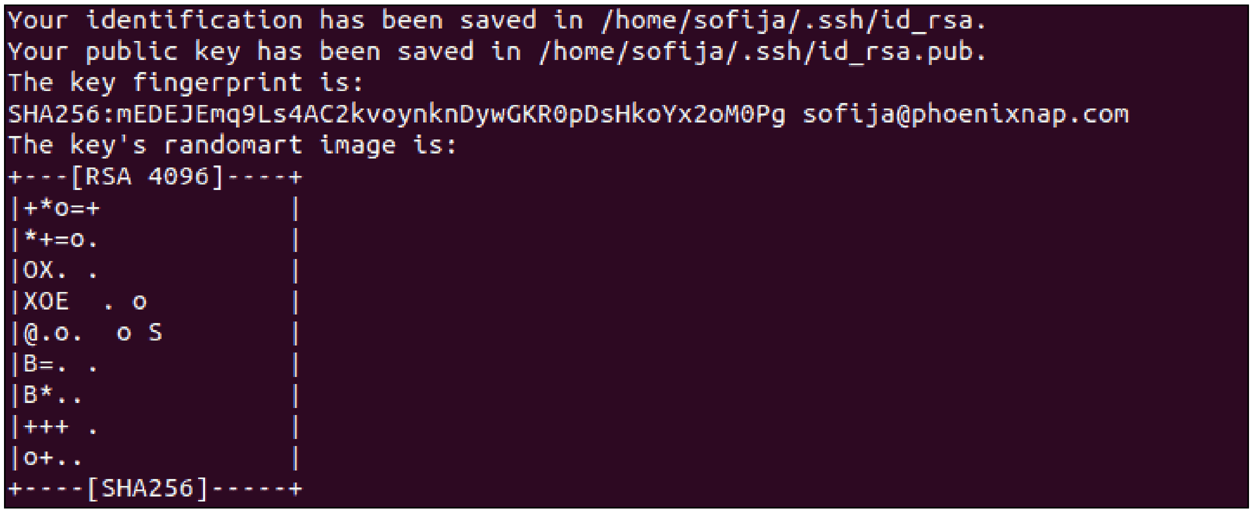
2. Next, type in the location where you want to store the keys or hit **Enter** to accept the default path.

3. It also asks you to set a passphrase. Although this makes the connection even more secure, it may interrupt when setting up automated processes. Therefore, you can type in a passphrase or just press **Enter** to skip this step.



```
Enter file in which to save the key (/home/sofija/.ssh/id_rsa):
Enter passphrase (empty for no passphrase):
```

4. The output then tells you where it stored the identification and public key and gives you the key fingerprint.

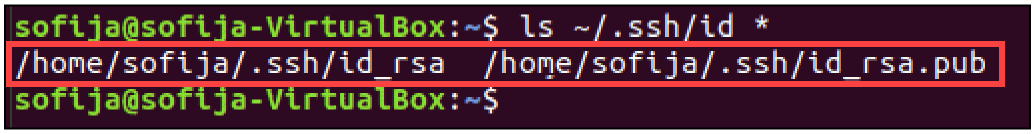


```
Your identification has been saved in /home/sofija/.ssh/id_rsa.
Your public key has been saved in /home/sofija/.ssh/id_rsa.pub.
The key fingerprint is:
SHA256:mEDEJEmq9Ls4AC2kvoynknDywGKR0pDsHkoYx2oM0Pg sofija@phoenixnap.com
The key's randomart image is:
+---[RSA 4096]----+
|+*o=+            |
|*+=o.            |
|OX. .            |
|XOE  . o         |
|@.o.  o S        |
|B=. .            |
|B*..             |
|+++ .            |
|o+..             |
+----[SHA256]-----+
```

5. Verify you have successfully created the SSH key pair by running the command:

```
ls -al ~/.ssh/id_*.pub
```

You should see the path of the identification key and the public key, as in the image below:



```
sofija@sofija-VirtualBox:~$ ls ~/.ssh/id *
/home/sofija/.ssh/id_rsa  /home/sofija/.ssh/id_rsa.pub
sofija@sofija-VirtualBox:~$
```

Step 2: Upload Public Key to Remote Server

You can upload the public SSH key to a remote server with the **ssh-copy-id** command or the **cat** command. Below you can find both options.

Option 1: Upload Public Key Using the ssh-copy-id Command

To enable passwordless access, you need to upload a copy of the public key to the remote server.

1. Connect to the remote server and use the **ssh-copy-id** command:

```
ssh-copy-ide [remote_username]@[server_ip_address]
```

2. The public key is then automatically copied into the **.ssh/authorized_keys** file.

Option 2: Upload Public Key Using the cat Command

Another way to copy the public key to the server is by using the **cat** command.

1. Start by connecting to the server and creating a **.ssh** directory on it.

```
ssh [remote_username]@[server_ip_address] mkdir -p .ssh
```

2. Then, type in the password for the remote user.

3. Now you can upload the public key from the local machine to the remote server. The command also specifies that the key will be stored under the name ***authorized_keys*** in the newly created **.ssh** directory:

```
cat .ssh/id_rsa.pub | ssh [remote_username]@[server_ip_address] 'cat >> .ssh/authorized_keys'
```

Step 3: Log in to Server Without Password

With the SSH key pair generated and the public key uploaded to the remote server, you should now be able to connect to your dedicated server without providing a password.

Check whether the setup works by running the command:

```
ssh [remote_username]@[server_ip_address]
```

The system should directly log you in to the remote server, no password required.

4. Advantages and Disadvantages of using SSH.
Ans:

Advantages of SSH:

- It is available free for non-commercial use
- The open-source version has gone through improvements like bug fixes, patches, and offers many additional functionalities.
- SSH may offer multiple services using the same connection
- SSH helps you to securely tunnel insecure applications like SMTP, IMAP, POP3, and CVS.
- The tunnelling of ports works effectively for simple VPNs.
- It offers strong authentication and secure communications over insecure channels.
- SSH allows users to log into another computer over an insecure network securely.
- Provide privacy of your data via strong encryption.
- The integrity of communications performed in such a way that it cannot been altered.
- Authenticate proof of identity of senders and receivers.
- Allows you to back or forward or to encrypt other TCP/IP- based sessions.
- Allows the user to view the contents of directories, edit files, and access custom database applications remotely.

<u>Disadvantage of SSH:</u>

- Telnet connection does not allow you to run GUI tools.
- It is not designed to transmit cursor movements or GUI movement information.
- It is not a secure protocol.
- SSH protocol not able to fix all TCP’s problems since TCP runs below SSH.
- SSH cannot protect users from attacks made through other protocols.
- This protocol does not protect Trojan horses or viruses.