Lab 14 – How to SSH into a server from a Linux machine

Task 1: Connect to a Remote Server via SSH

Logic / Purpose:

This task introduces you to the basic usage of SSH (Secure Shell), which allows you to remotely access and manage a device over a secure network connection. SSH ensures encrypted communication between your local machine and the remote server.

Detailed Steps:

1. **Open the terminal** on your Linux system.

Use the SSH command to initiate a connection to the remote server:

```
ssh osboxes@192.168.1.221
```

2.
   - `ssh`: The command to start the Secure Shell connection.
   - `osboxes`: The username you want to log in as on the remote server.
   - `192.168.1.221`: The IP address of the remote server.

Accept the server's fingerprint when prompted with:

```
The authenticity of host '192.168.1.221 (192.168.1.221)' can't be established.
Are you sure you want to continue connecting (yes/no)?
```

3.
   - Type `yes` and press Enter. This adds the server's key to your list of known hosts.

Enter the password when prompted. Use:

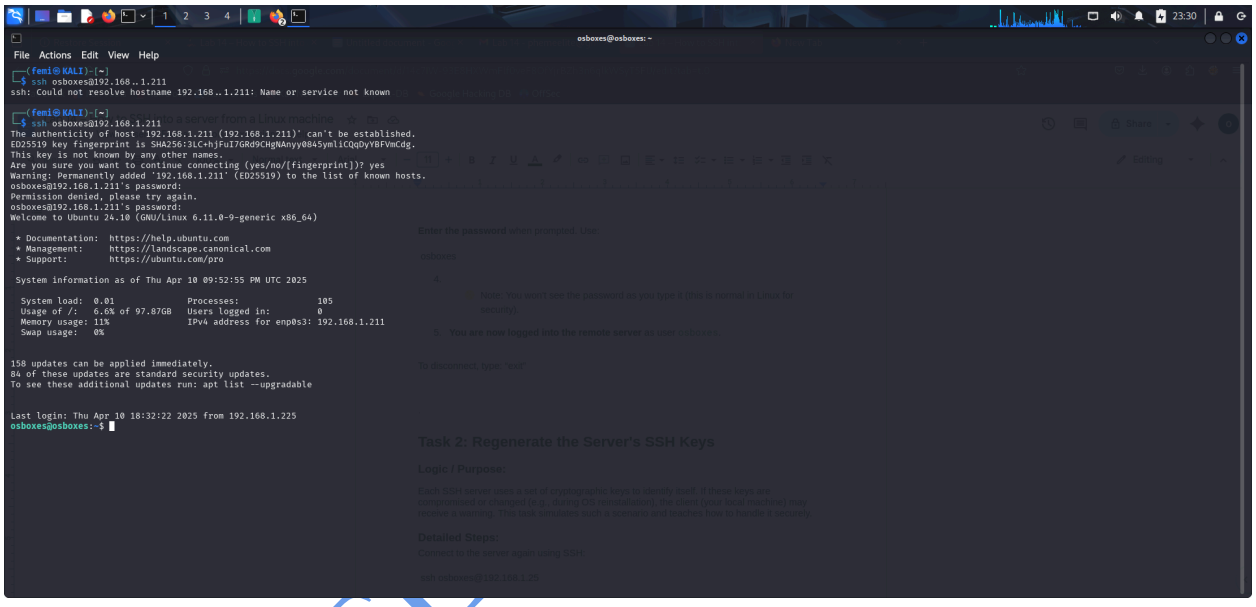
osboxes

4.

   🌕 Note: You won't see the password as you type it (this is normal in Linux for security).

5. **You are now logged into the remote server** as user `osboxes`.

To disconnect, type: “exit”



Task 2: Regenerate the Server's SSH Keys

Logic / Purpose:

Each SSH server uses a set of cryptographic keys to identify itself. If these keys are compromised or changed (e.g., during OS reinstallation), the client (your local machine) may receive a warning. This task simulates such a scenario and teaches how to handle it securely.

Detailed Steps:

Connect to the server again using SSH:

ssh osboxes@192.168.1.221

1.

Gain root privileges:

sudo su -

2.

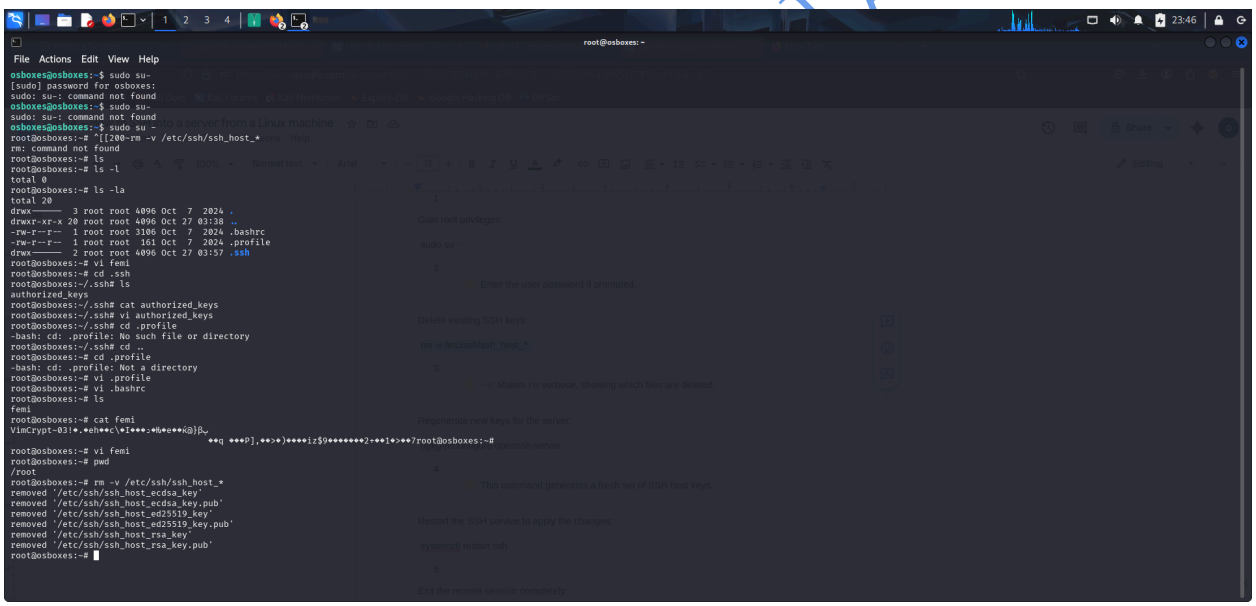
- Enter the user password if prompted.

Delete existing SSH keys:

rm -v /etc/ssh/ssh_host_*

3.

- `-v`: Makes `rm` verbose, showing which files are deleted.

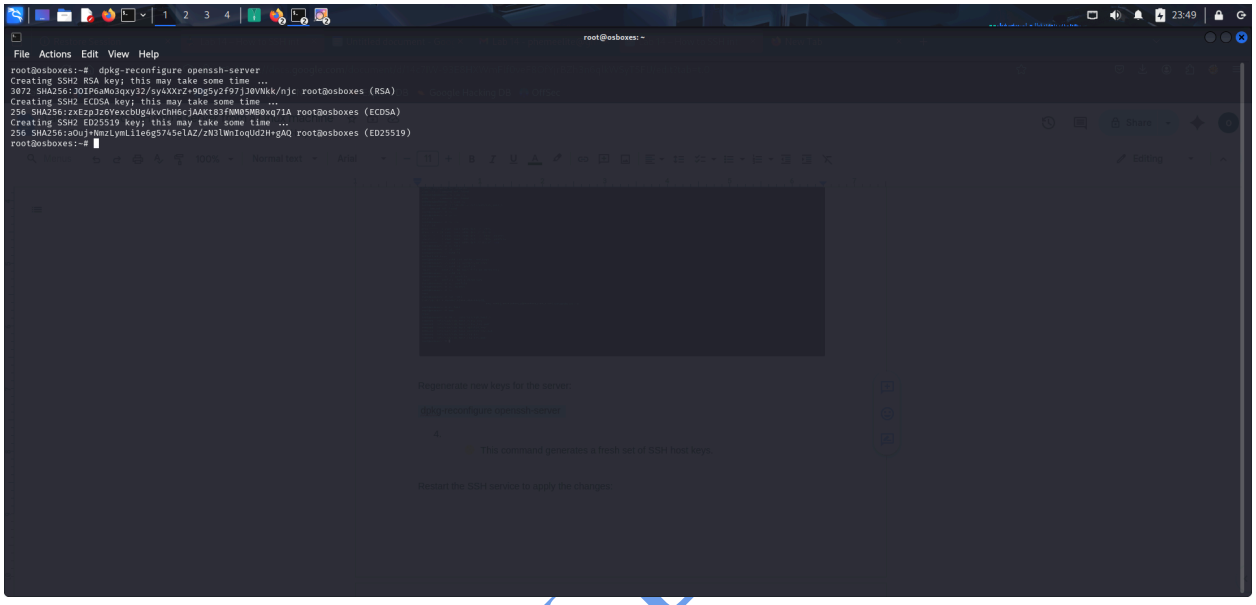


Regenerate new keys for the server:

dpkg-reconfigure openssh-server

4.

🌕 This command generates a fresh set of SSH host keys.



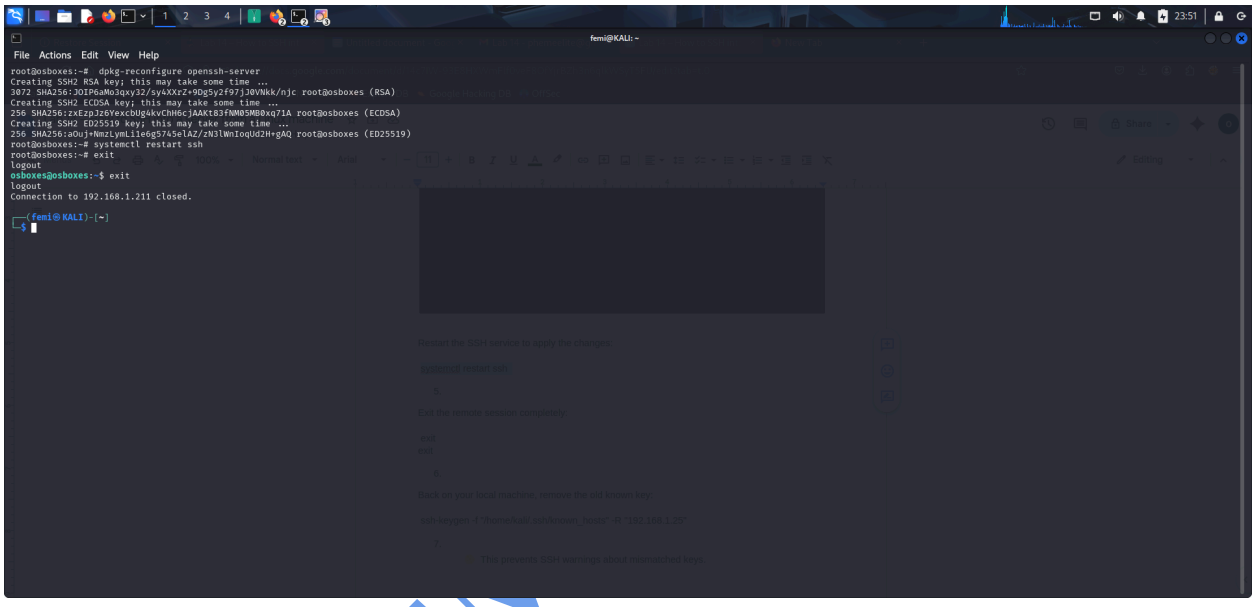
Restart the SSH service to apply the changes:

systemctl restart ssh

5.

Exit the remote session completely:

“exit”
“exit”



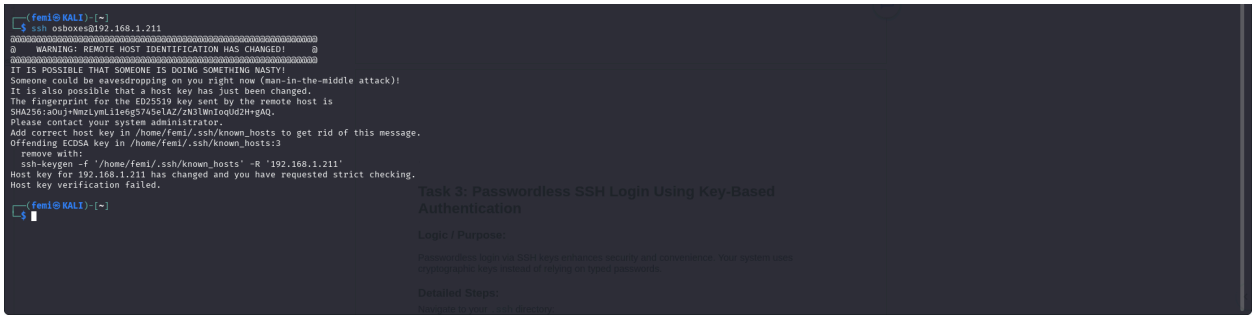
6.

🌕 This prevents SSH warnings about mismatched keys.

Reconnect using SSH and accept the new key:

ssh osboxes@192.168.1.221

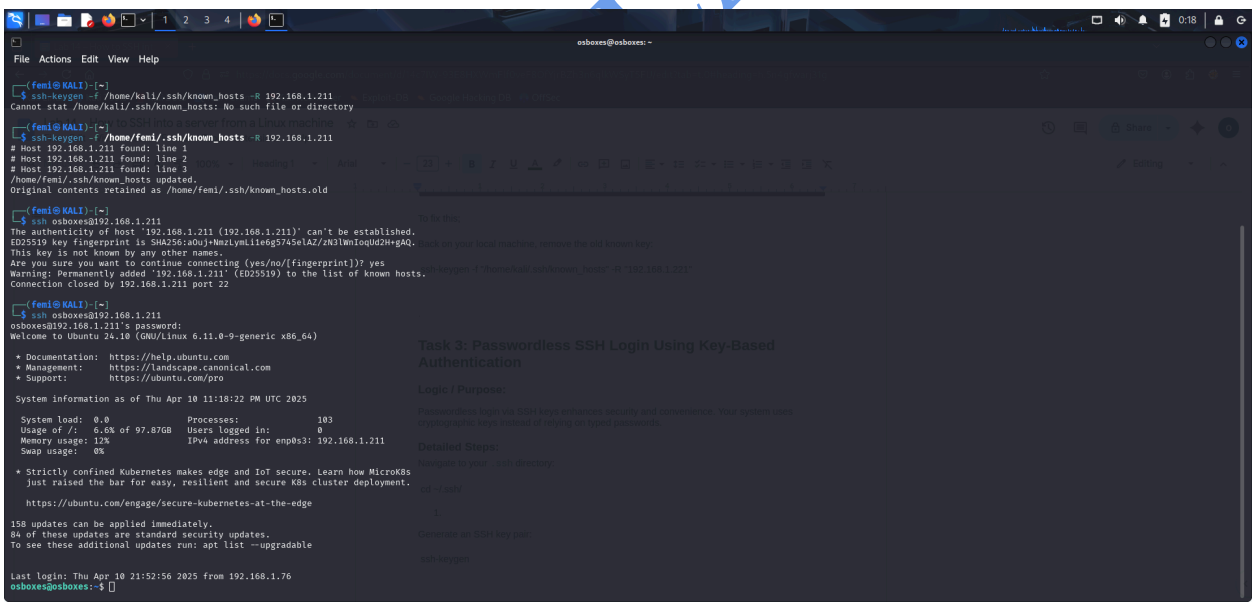
It would throw a security warning: “WARNING : REMOTE IDENTIFICATION HAS CHANGED”



To fix this;

Back on your local machine, remove the old known key:

ssh-keygen -f "/home/kali/.ssh/known_hosts" -R "192.168.1.221"



Try SSH again:

`ssh osboxes@192.168.1.221`

You’ll be asked to trust the new key — type `yes`, and then log in as before.

Task 3: Passwordless SSH Login Using Key-Based Authentication

Logic / Purpose:

Passwordless login via SSH keys enhances security and convenience. Your system uses cryptographic keys instead of relying on typed passwords.

Detailed Steps:

Navigate to your `.ssh` directory:

```
cd ~/.ssh/
```

1.

Generate an SSH key pair:

```
ssh-keygen
```

2.
    - Press Enter to accept default filename.
    - Press Enter again to leave the passphrase blank (or enter a passphrase for extra security).

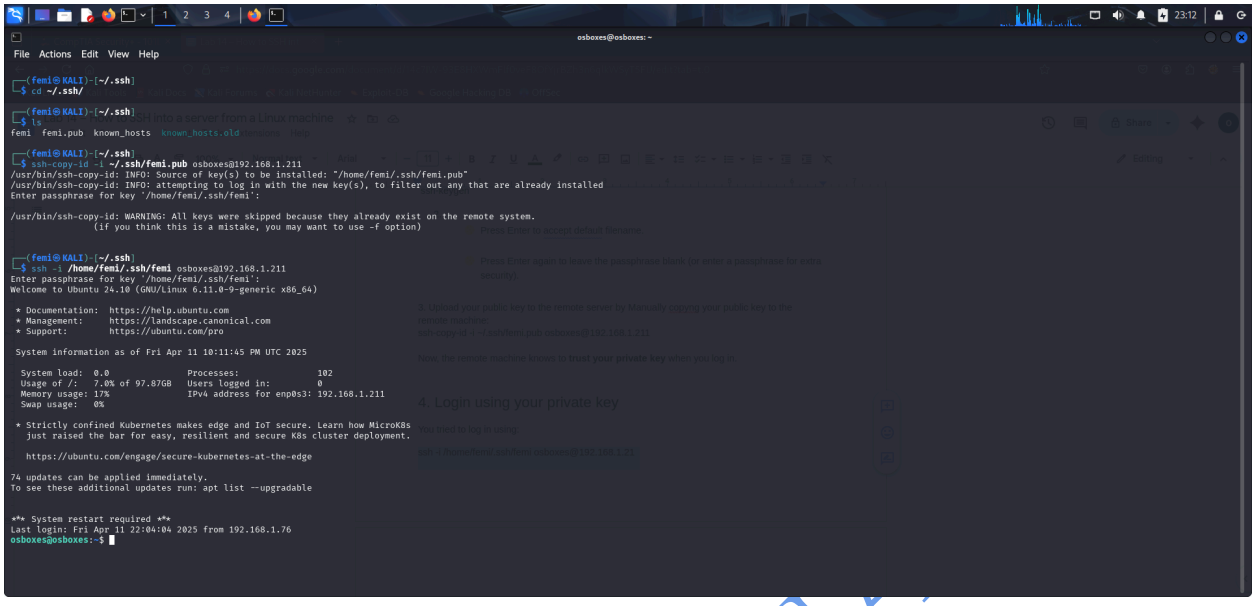
3. Upload your public key to the remote server by Manually copyng your public key to the remote machine:
ssh-copy-id -i ~/.ssh/femi.pub osboxes@192.168.1.211

Now, the remote machine knows to **trust your private key** when you log in.

4. Login using your private key

You tried to log in using:

ssh -i /home/femi/.ssh/femi osboxes@192.168.1.21



(Optional) Disable password authentication for added security:

Disable SSH Password Login for Better Security

🎯 Purpose:

To enhance the security of your server by disabling password logins and requiring SSH key-based authentication only. This prevents brute-force password attacks and ensures only devices with your private key can log in.

✅ Step 1: SSH into your server using your SSH key

bash

CopyEdit

```
ssh -i ~/.ssh/femi osboxes@192.168.1.211
```

🧠 Why?

You need to log into the server before you can change its configuration. This command logs you in using your SSH private key (`femi`), not a password.

✅ Step 2: Open the SSH server configuration file

```
sudo nano /etc/ssh/sshd_config
```

🧠 Why?

This file controls how the SSH server behaves — what it allows and disallows. We'll change it so the server no longer accepts passwords for login.

> **`sudo` gives you administrator rights, and `nano` is the text editor we're using.**

✅ Step 3: Find the `# Authentication:` section

Scroll until you find this (or something similar):

```
# Authentication:

#LoginGraceTime 2m

#PermitRootLogin prohibit-password

#StrictModes yes

#MaxAuthTries 6

#MaxSessions 10
```

Right below this section, add the following line (or find and modify it if it already exists):

```
PasswordAuthentication no
```

🧠 Why?

This tells the SSH server:

> "Do not accept password logins anymore. Only allow people with a valid SSH key."

Make sure this line is not commented out (no `#` at the beginning).

✅ Step 4: Save and exit the file

In `nano`, do this:

- Press `Ctrl + O` → saves the file
- Press `Enter` → confirms the filename
- Press `Ctrl + X` → exits the editor

🧠 Why?

You've made a change to the SSH configuration — now you need to save and close it so you can apply it.

✅ Step 5: Restart the SSH service

bash

```
sudo systemctl restart ssh
```

🧠 Why?

This restarts the SSH server so that your new settings (disabling password login) take effect immediately.

✅ Step 6: Test the new login behavior from your Kali machine

✔️ This should still work (because it uses your private key):

bash

```
ssh -i ~/.ssh/femi osboxes@192.168.1.211
```

❌ This should now fail (because password login is disabled):

bash

```
ssh osboxes@192.168.1.211
```


Task 4: SSH Port Forwarding (Tunneling)

Logic / Purpose:

SSH tunneling allows you to securely access a network service (e.g., a web server) on another machine through the remote server.

Detailed Steps:

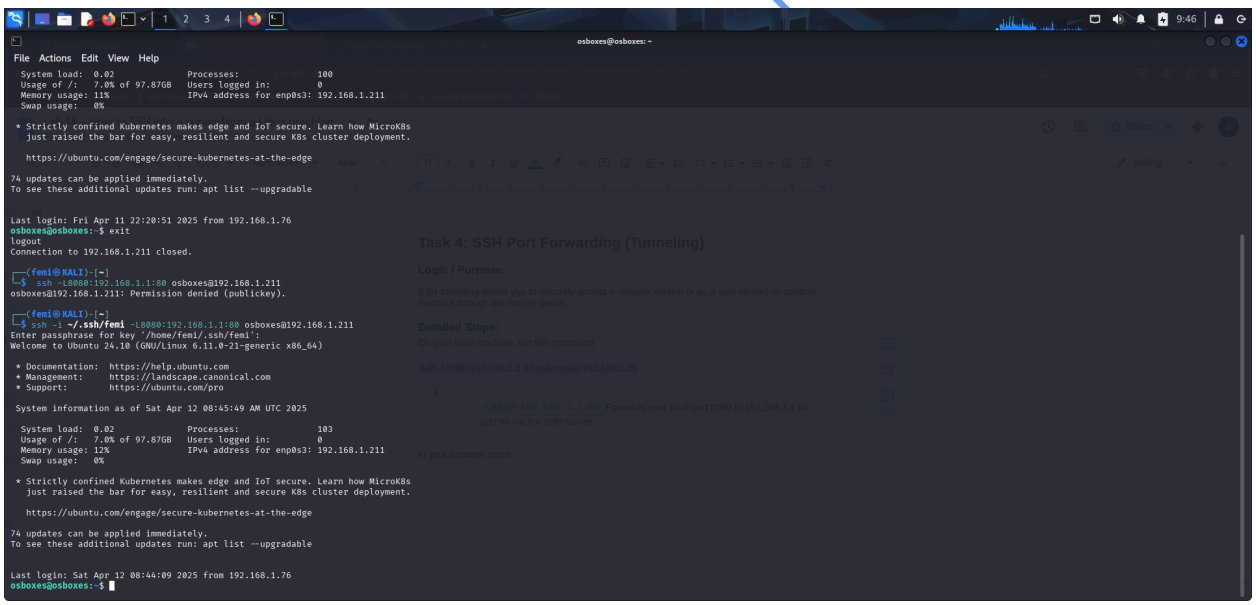
On your local machine, run this command:

ssh -L8080:192.168.1.1:80 osboxes@192.168.1.211

1.
   - `-L8080:192.168.1.1:80`: Forwards your local port 8080 to 192.168.1.1 on port 80 via the SSH server.

Note: Incase you already disabled SSH password for login, the the above command won't work. So instead, use:

ssh -i ~/.ssh/femi -L8080:192.168.1.1:80 osboxes@192.168.1.211

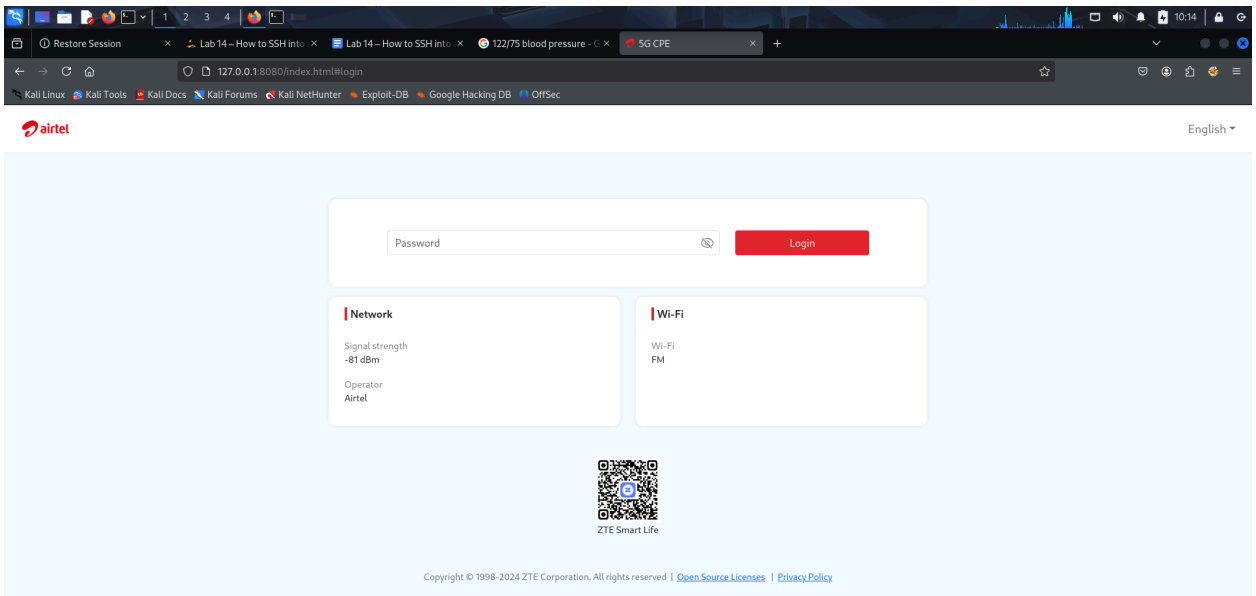


In your browser, open:

http://127.0.0.1:8080

2.

- This accesses the internal service via the SSH tunnel.



Task 5: Secure File Copy Using SCP

Logic / Purpose:

SCP (Secure Copy) allows you to copy files from your local system to a remote server (or vice versa) over an SSH connection.

Detailed Steps:

Copy a local file to the remote server:

scp /etc/hosts osboxes@192.168.1.25:/tmp/

1.

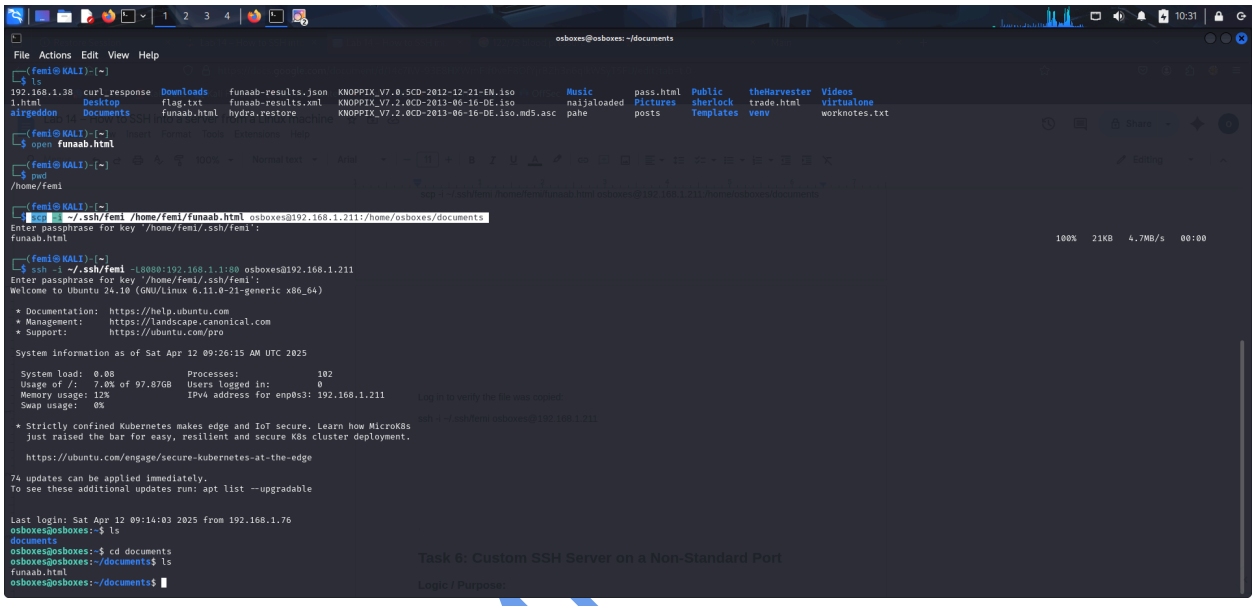
- This command sends the `/etc/hosts` file to `/tmp/` on the remote server.

Note: Incase you already disabled SSH password for login, the the above command won't work. So instead, use:

scp -i ~/.ssh/femi /home/femi/funaab.html osboxes@192.168.1.211:/home/osboxes/documents

Log in to verify the file was copied:

ssh -i ~/.ssh/femi osboxes@192.168.1.211



Task 6: Custom SSH Server on a Non-Standard Port

Logic / Purpose:

This demonstrates how to run a custom SSH server on a different port without needing root privileges—useful in restricted environments or for testing.

Detailed Steps:

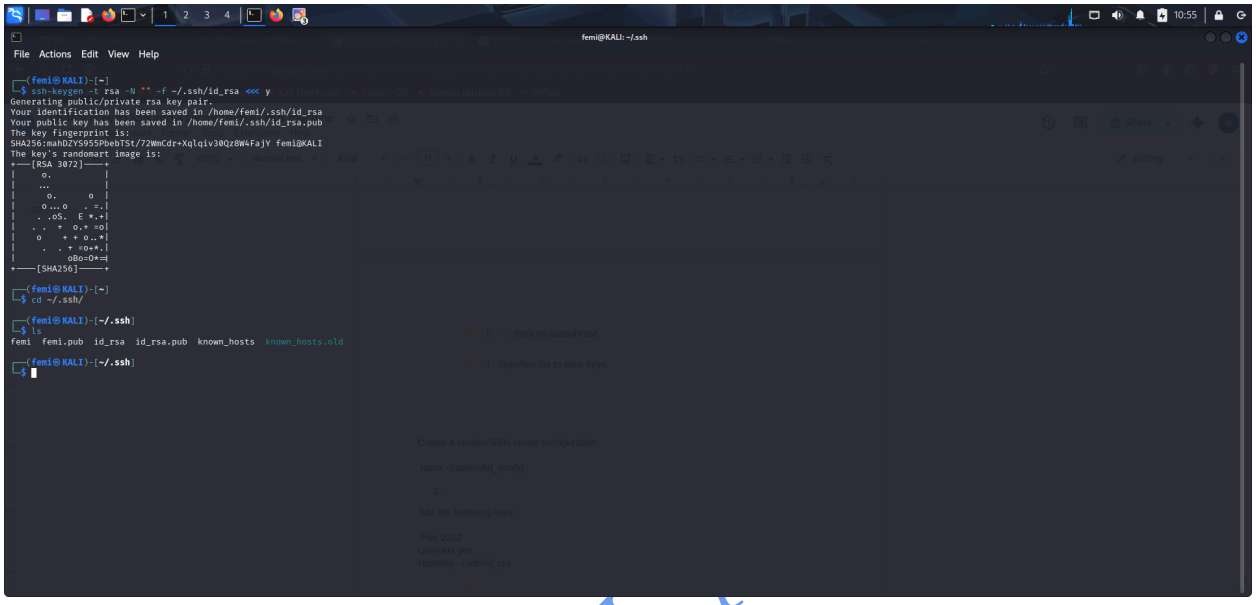
Generate a new SSH key pair:

ssh-keygen -t rsa -N "" -f ~/.ssh/id_rsa <<< y

1.

- `-t rsa`: Specifies RSA key type.

- `-N ""`: Sets no passphrase.
- `-f`: Specifies file to save keys.



Create a custom SSH server configuration:

nano ~/.ssh/sshd_config

2.

Add the following lines:

Port 2222
UsePAM yes
HostKey ~/.ssh/id_rsa

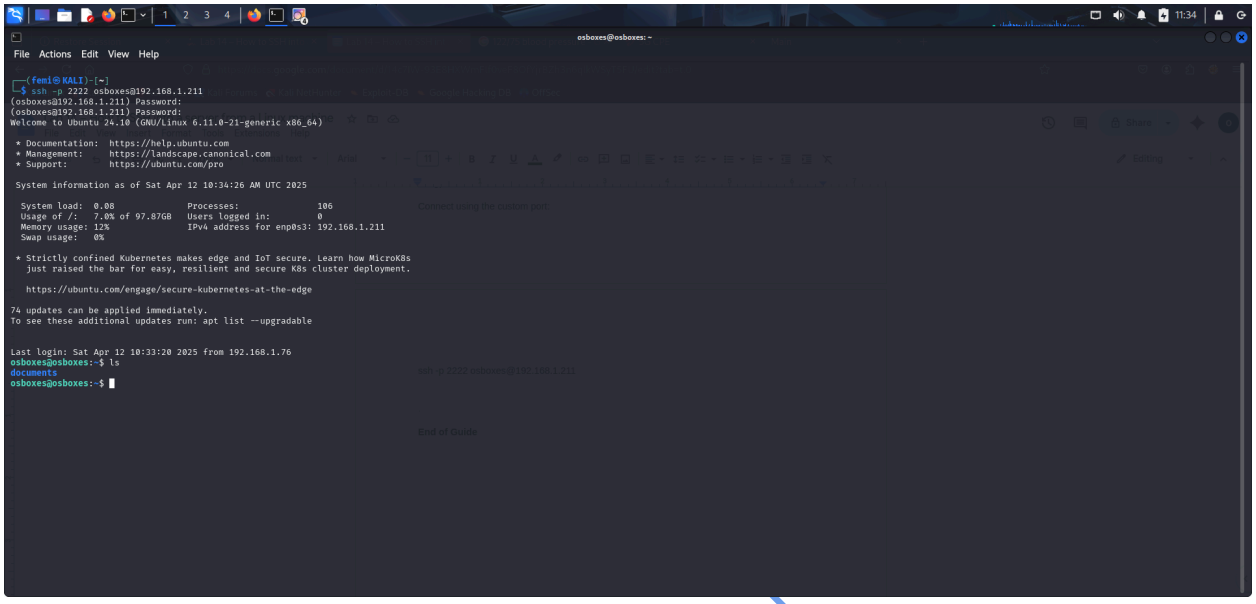
Start the SSH server with this config:

/usr/sbin/sshd -f ~/.ssh/sshd_config

3.

Connect using the custom port:

ssh -p 2222 osboxes@192.168.1.211



End