How to Secure a Personal Web Server

With:
Anjola F

In today's digital landscape, securing your web server is crucial to protecting sensitive data, preventing unauthorized access, and ensuring a smooth online experience.

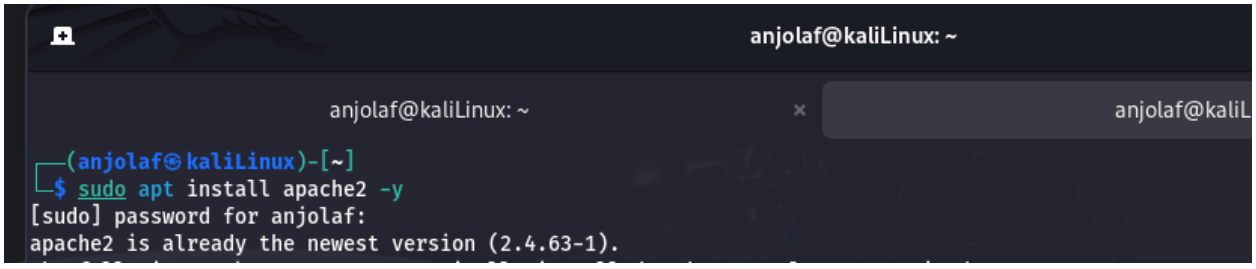
Walk we me as I show you how I set up an Apache web server and secured it with SSL/TLS while implementing firewall and access controls.

Step 1: System Update

I updated my Kali Linux Machine by running **sudo apt update && sudo apt upgrade -y** to ensure I have the latest security patches thereby reducing the risks of getting exploited by attackers.

Step 2: Install, Start and Enable Apache

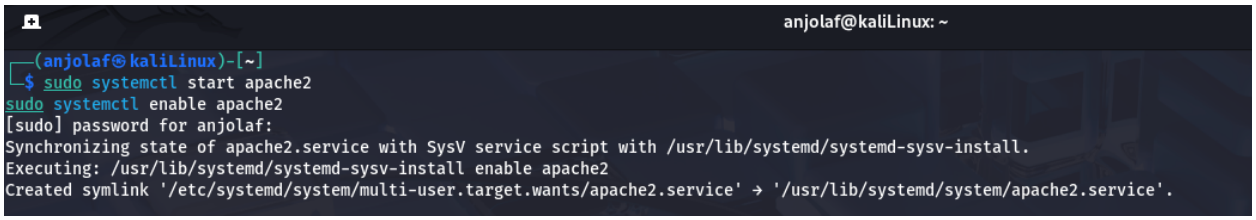
I installed Apache by running **sudo apt install apache2 -y** in Kali.



Then, I started and enabled Apache by running the following:

sudo systemctl start apache2

sudo systemctl enable apache2



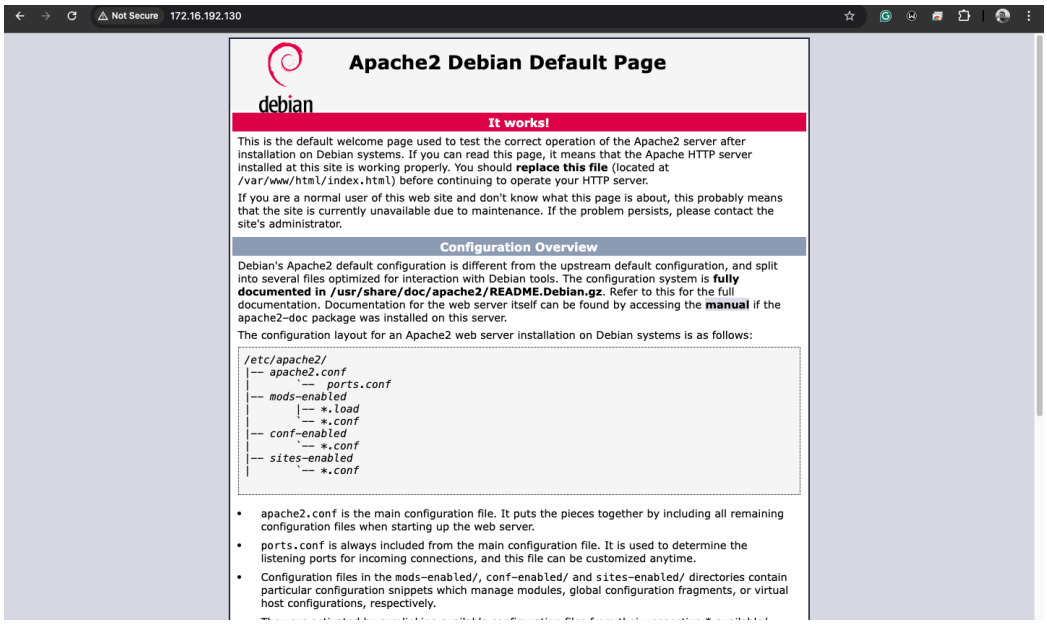
Step 3: Allow Web Traffic Through the Firewall

To help block unauthorized access while allowing legitimate traffic (HTTP/HTTPS) to the server, I ran the following to allow and enable UFW:

sudo ufw allow 'Apache Full'
sudo ufw enable

Step 4: Test Web Server

Before proceeding with security enhancements, I had to ensure that I was able to access my Web server at http://http://172.16.192.130/

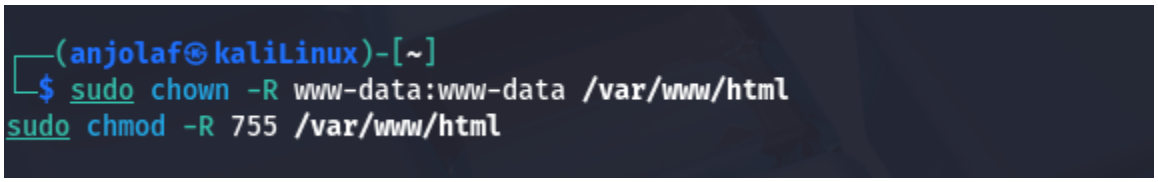


Apache Welcome Page

Step 5: Securing The Web Server

I started by changing the default web root permissions to prevent unauthorized users from modifying web files and protecting against attacks like website defacement.

sudo chown -R www-data:www-data /var/www/html
sudo chmod -R 755 /var/www/html



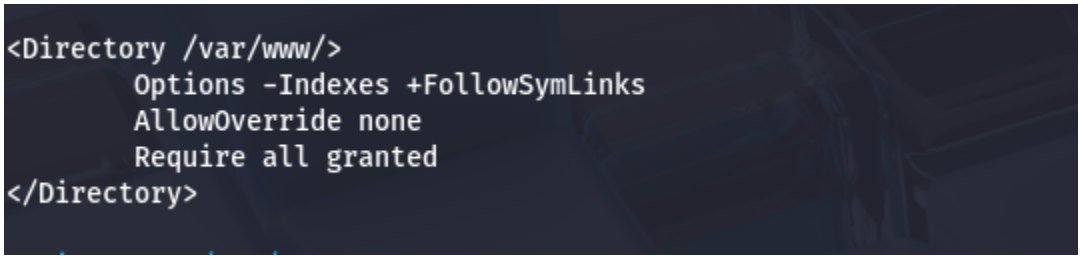
Step 6: Disabled Directory Listing

I disabled Directory Listing by editing the Apache Config file in Kali:

sudo nano /etc/apache2/apache2.conf

Under <Directory /var/www/>
    Options -Indexes +FollowSymLinks
    AllowOverride None
    Require all granted
</Directory>

I edited the Options Index and FollowSymLinks Inputs to prevent attackers from viewing all files in my web directory, which could expose sensitive information.



Step 7: Secure SSH Access

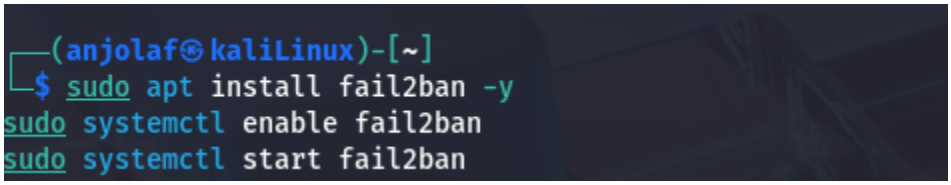
I changed the PermitRootLogin to NO and edited the port number to port 2000 to make it harder for attackers to brute-force my credentials.

sudo nano /etc/ssh/sshd_config

Then I restarted SSH with **sudo systemctl restart ssh**

Step 8: Enable Fail2Ban

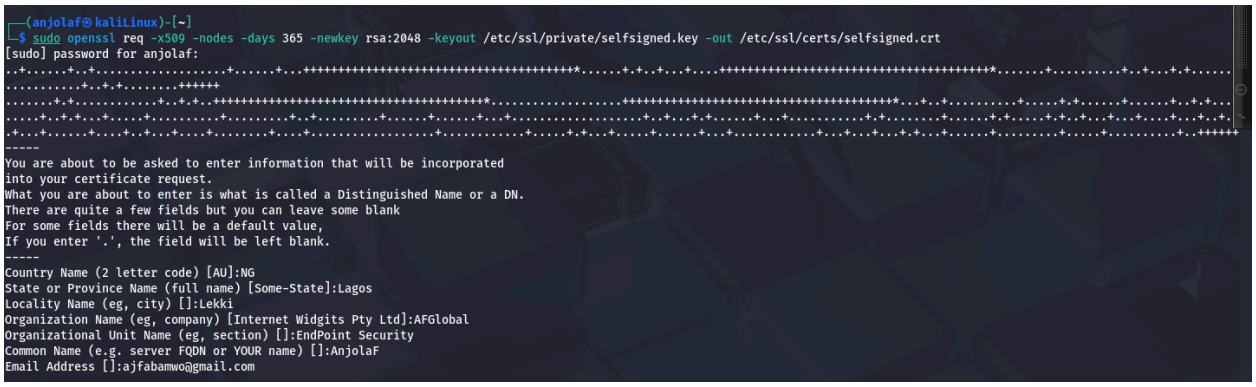
I enabled and Installed Fail2Ban to block IPs that repeatedly fail to log in, preventing brute-force attacks.



Configure SSL/TLS

At the time I wrote this, I did not have access to a domain so I generated a Self Signed Certificate for local use only by running:

sudo openssl req -x509 -nodes -days 365 -newkey rsa:2048 -keyout /etc/ssl/private/selfsigned.key -out /etc/ssl/certs/selfsigned.crt



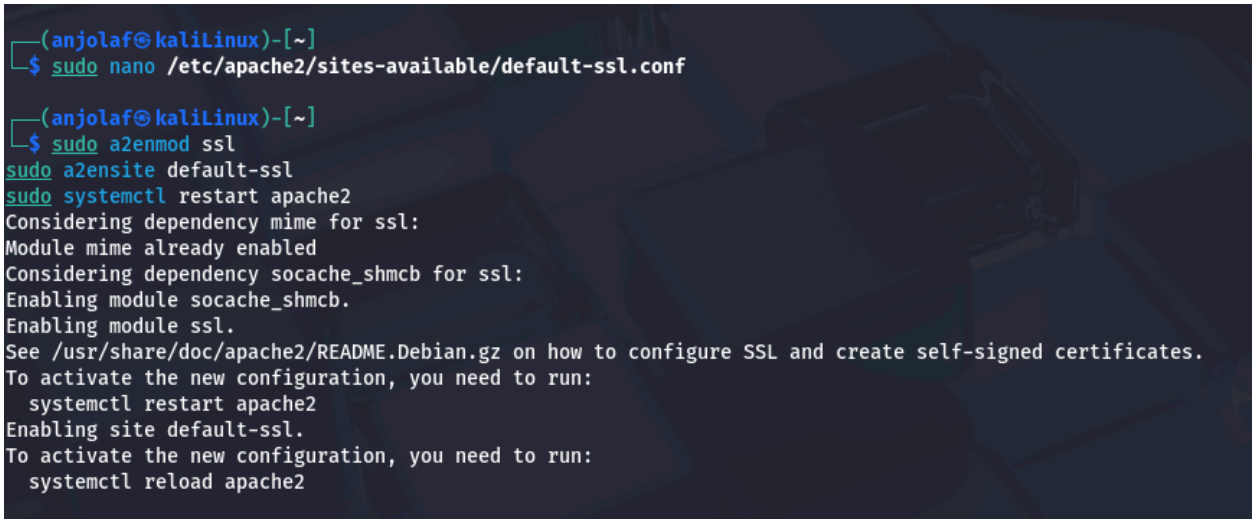
Then, I configured Apache to use the Self Signed Certificate by running:
sudo nano /etc/apache2/sites-available/default-ssl.conf

And ensuring these lines were present in the config file:

SSLCertificateFile /etc/ssl/certs/selfsigned.crt
SSLCertificateKeyFile /etc/ssl/private/selfsigned.key

I then ran the following to Enable the SSL Module and Restart Apache.

sudo a2enmod ssl
sudo a2ensite default-ssl
sudo systemctl restart apache2



This method allows me to use HTTPS locally without needing a public domain.

In Conclusion, I was able to:
✔ Install **Apache** web server.
✔ Configure a **firewall** to allow only necessary traffic.
✔ Secure server with **SSH hardening, Fail2Ban, and correct file permissions**.
✔ Configure **Self-Signed SSL** for HTTPS security.
✔ Enforce **automatic HTTPS redirection** to protect data transmission.