This PDF Lecture is made by
Abid Biswas, Ethical Hacker **(Purple Teaming)**
(CEH, NSE1, NSE2, NSE3,
$ISC^2(CC)$, *Google Cyber Security Professional Cert*)
BSc in Information and Communications Engineering (ICE) at EWU

To install Apache2 Web Server on Ubuntu Linux Platform:

We can even install Apache web server on windows

sudo apt-get install apache2

What's going to happen is it's going to reach out to any package repositories configured for this specific machine and pull that down over the Internet. Now naturally, from a security standpoint, web application security begins at the foundation. It means making sure the operating system, in this case, Ubuntu Linux, is kept up-to-date, and configured securely, including install packages being updated like the Apache2 web server.

So, what we can do is we can always run sudo apt-get update to make sure that our list of package repositories is up to date and you don't have to point to the Internet. Your organization if it uses Linux extensively, might have local private repositories with carefully curated software packages that are made available to install on Linux hosts internally. We can also run the sudo apt-get upgrade command to upgrade.

~$ sudo apt-get update

$ sudo apt-get upgrade

So, we know that this is an important part of web application security, at this level, the operating system and web server stack level.

So, the next thing to do then is to make sure that the Apache web server, daemon or background service is up and running. I can do that with sudo service, it's called apache2.

~$ sudo service apache2 status

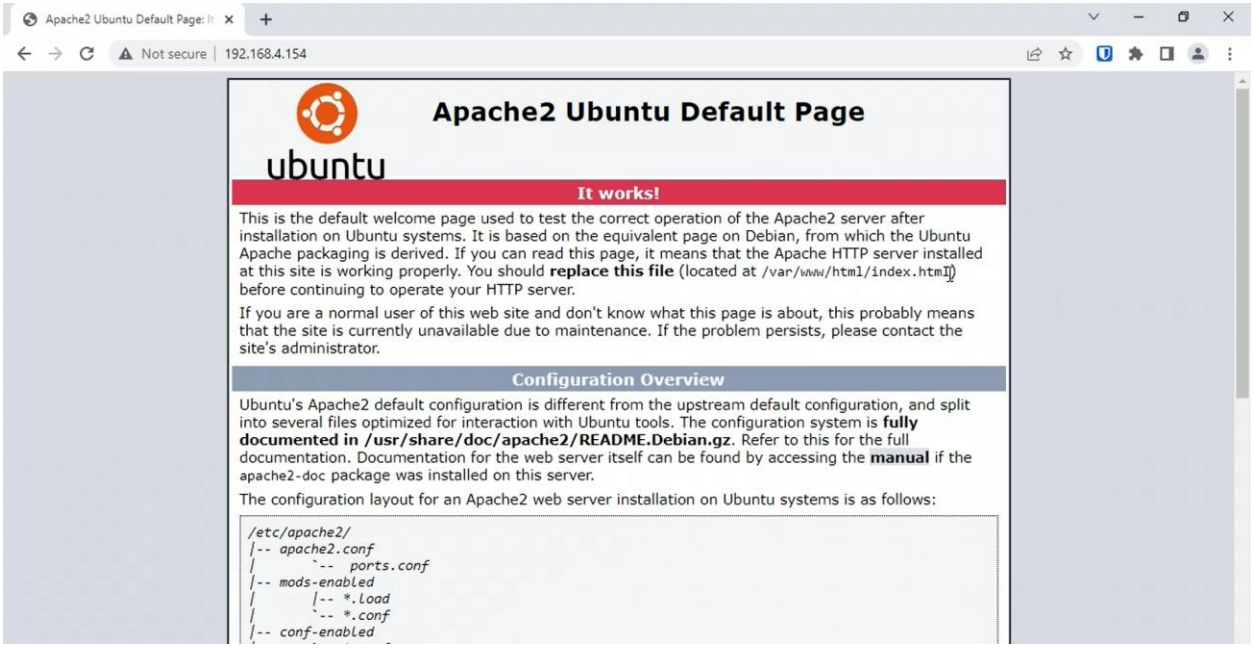
```
cblackwell@ubuntu1:~$ sudo service apache2 status
● apache2.service - The Apache HTTP Server
     Loaded: loaded (/lib/systemd/system/apache2.service; enabled; vendor pre>
     Active: active (running) since Sun 2022-04-24 16:25:07 UTC; 8min ago
       Docs: https://httpd.apache.org/docs/2.4/
   Main PID: 2112 (apache2)
      Tasks: 55 (limit: 4575)
     Memory: 7.8M
     CGroup: /system.slice/apache2.service
             ├─2112 /usr/sbin/apache2 -k start
             ├─2113 /usr/sbin/apache2 -k start
             └─2114 /usr/sbin/apache2 -k start

Apr 24 16:25:07 ubuntu1 systemd[1]: Starting The Apache HTTP Server...
Apr 24 16:25:07 ubuntu1 apachectl[2109]: AH00558: apache2: Could not reliably>
Apr 24 16:25:07 ubuntu1 systemd[1]: Started The Apache HTTP Server.
lines 1-15/15 (END)
```

and then I'll pass it the status parameter. So, we get a return to listing that shows us that the Apache2 web server daemon is active and up and running, so the Apache HTTP server then has been started. **Press Q**

for quit. I'm going to run **~$ ip a** , IP address which will show me my local loopback IP along with any other IPv6 and IPv4 addresses for this host. For example, I'm interested in the IPv4 address 192.168.4.154.

```
cblackwell@ubuntu1: ~
cblackwell@ubuntu1:~$ ip a
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue state UNKNOWN group defa
ult qlen 1000
    link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
    inet 127.0.0.1/8 scope host lo
       valid_lft forever preferred_lft forever
    inet6 ::1/128 scope host
       valid_lft forever preferred_lft forever
2: ens33: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc fq_codel state UP g
roup default qlen 1000
    link/ether 00:0c:29:d2:6a:4e brd ff:ff:ff:ff:ff:ff
    inet 192.168.4.154/22 brd 192.168.7.255 scope global dynamic ens33
       valid_lft 13753sec preferred_lft 13753sec
    inet6 fe80::20c:29ff:fed2:6a4e/64 scope link
       valid_lft forever preferred_lft forever
3: docker0: <NO-CARRIER,BROADCAST,MULTICAST,UP> mtu 1500 qdisc noqueue state D
OWN group default
    link/ether 02:42:0f:ba:5d:b4 brd ff:ff:ff:ff:ff:ff
    inet 172.17.0.1/16 brd 172.17.255.255 scope global docker0
       valid_lft forever preferred_lft forever
cblackwell@ubuntu1:~$
```

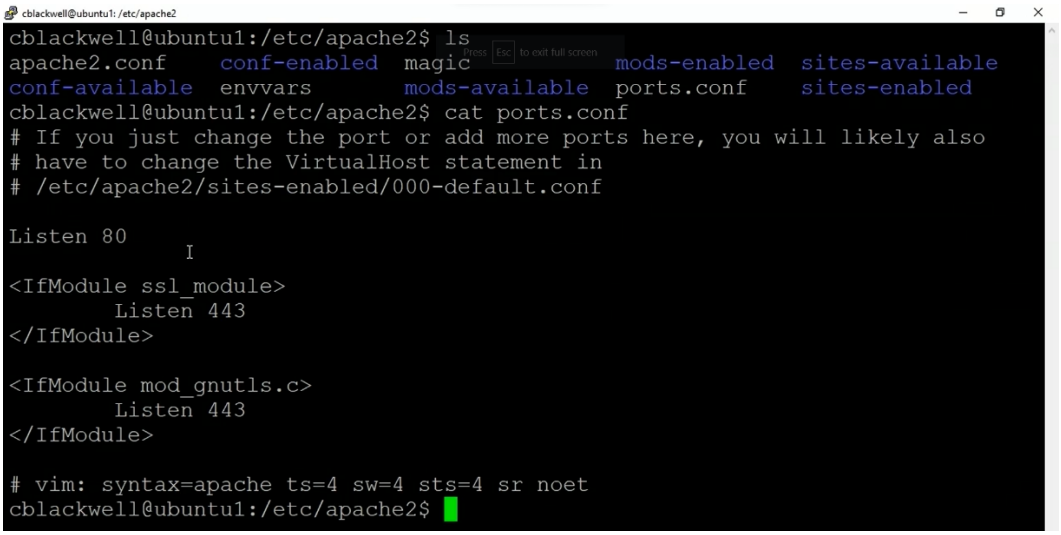
Why don't we open up a web browser on this network on a different host and just try to connect to the Apache web page? So, when I do that, it opens up the Apache2 Ubuntu default web page, and it says it works. So, we know then that the default web page configuration for that web server is now up and running on our Ubuntu Linux host.



And, of course, we've also patched it to make sure we've got the latest updates applied. Now, the great thing about a lot of these web server default web page is they also serve double duty as not only a test page, but

also a configuration page where it gives you some examples of what to do. Here **it shows us the configuration file layout on this host for the Ubuntu Apache2 web server, such as going into /etc/apache2**, let's explore that a little bit.

So, back here in Linux, I'm going to change directory to /etc/apache2. I'll clear the screen, I'll do an ls. So, we've got the apache2.conf file along with the ports.conf file.



it defines the listening ports for the web server. **Here it's listening on port 80, if we've got the ssl_module enabled for the web server with the PKI certificate, then it's going to listen by default on port 443, but we know that SSL really is deprecated, should never be used. Instead, we should be using TLS, Transport Layer Security. Now, if the ssl_module is not enabled for the web server stack, then it won't be listening for that type of connection on 443 using the SSL network security protocol. So, we could even comment out those lines if we wanted to. But really, as long as we don't enable the ssl_module, we're good.**

Now speaking of that, why don't we take a look at the apache2 conf file, the config file, the main config file for the web server. So, I'll clear the screen, I'm actually going to use the nano editor to open it up and it's called apache2.conf.

: /etc/apache2$ nano apache2.conf

Comments are notated on the line with a hashtag symbol or a pound symbol. That means that line is a comment, it is thought to be interpreted as anything such as a configuration directive. So, notice for example, the ServerRoot directive is commented out, the default is /etc/apache2.

ServerRoot "/etc/apache2

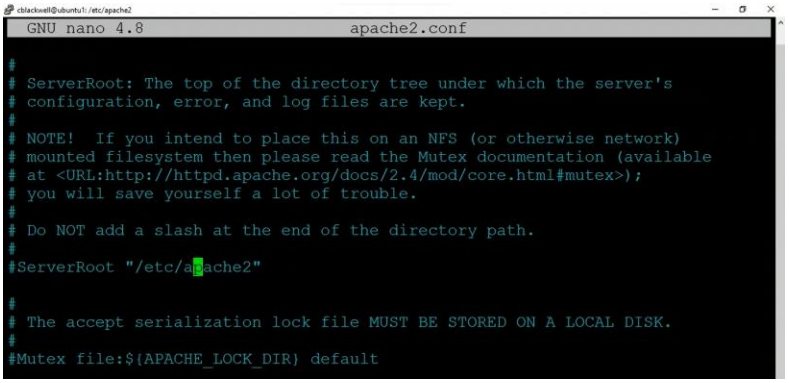
We then have the ErrorLog directive, which is using a built-in APAHE_LOG-DIR variable /error.log as the default error log unless otherwise specified for the Apache web server.

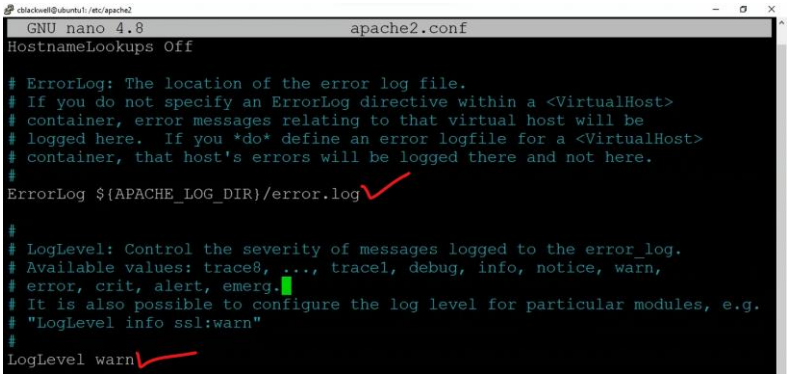
ErrorLog ${APACHE_LOG_DIR}error.log

IncludeOptional mods-enabled/* . load

IncludeOptional mods-enabled/* . conf

The LogLevel is set to warning. Then we've got an Include module section for additional modules that provide additional functionality for the web server.





emerg: This is the most severe level, indicating an emergency condition where the server cannot continue to run.

alert: A condition that should be corrected immediately, but the server can still continue to run.

crit: A critical condition that might require immediate attention.

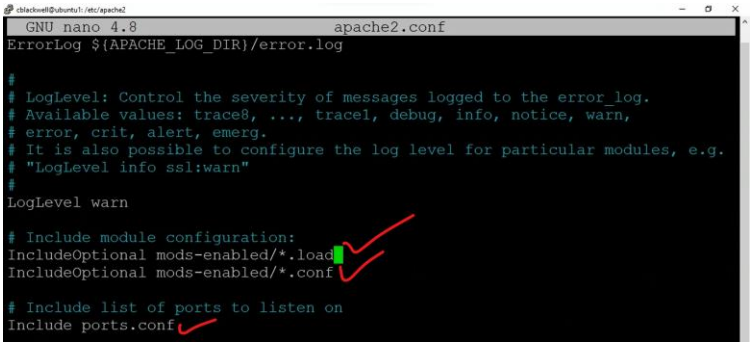
error: A general error condition.

warn: A warning message, indicating a potential problem that does not prevent the server from running.

notice: A notice message, providing general information about server activity.

info: Informational messages about the server's operation.

debug: Detailed debugging information for developers.



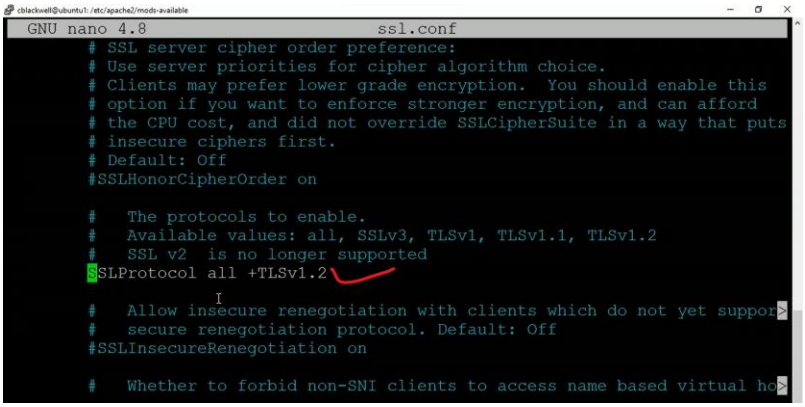
Then we've got an Include module section for additional modules that provide additional functionality for the web server. All the files that end with .load or .conf in the mods enabled location, which would determine things like whether SSL or TLS is being used. Whether we have some kind of Java server-side components that are up and running and notice it's also including the ports.conf file that we just looked at. As we go further down through the apache2.conf file, we also have some options for what permissions are allowed for different subdirectories on the web server. So I'm going to **press Ctrl+X to close out of that file**. We haven't made any changes.

ls

And if I were to do an ls in there, we've got all kinds of files that end with either .conf or .load as per our apache2.conf file. For example, we have an ssl.conf and an ssl.load file.

So, let's open up the ssl.conf file, notice there's not a TLS file. These days, there's a lot of old references to SSL, even though we would be using its superseding network security protocol TLS, so I'm going to open up the ssl.conf file. I'm going to use the sudo prefix, so I can write to it. So I'm going to use the nano text editor built into Ubuntu Linux, ssl.conf.

/etc/apache2/mods-available$ sudo nano ssl.conf



What I'm interested in doing is making sure that the only protocol that is allowed in my particular case is TLSv1.2. Now notice the SSLProtocol directive here currently has all and the + means we want to also add support for TLSv1.2. You could also alternatively change that to - if you wanted to remove something or I could just say SSLProtocol. **I'm not going to use the word all, but instead +TLSv1.2 to make sure only that network security protocol version is to be used for this particular web server**. Now, syntactically, you could also just remove the plus sign, which implies that we only want to support TLSv1.2 on this host. Just bear in mind that if we want to support a newer version, such as TLSv1.3, we have to bear in mind that our configuration will have to change because we'll have to reference it in the ssl.conf file.

So, I'm going to go ahead and press Ctrl+X it asks to save the modified buffer I'll type in the letter Y I want to overwrite the same file, I'll just press Enter and it's done. At least the config is done, but it's not in effect until I restart the Apache2 web server daemon. Which I'll do with this you sudo service apache2 restart. I'll just use the Up arrow key to bring up that previous command, and I'm going to change restart to status.