LFS211

Linux Networking and Administration

Version 1.0

LFS211: Version 1.0

Published by:

the **Linux Foundation**
http://www.linuxfoundation.org

Contents

Chapter 1

Introduction

1.1 Labs

Exercise 1.1: Configuring the System for sudo

It is very dangerous to run a **root shell** unless absolutely necessary: a single typo or other mistake can cause serious (even fatal) damage.

Thus, the sensible procedure is to configure things such that single commands may be run with superuser privilege, by using the **sudo** mechanism. With **sudo** the user only needs to know their own password and never needs to know the root password.

If you are using a distribution such as **Ubuntu**, you may not need to do this lab to get **sudo** configured properly for the course. However, you should still make sure you understand the procedure.

To check if your system is already configured to let the user account you are using run **sudo**, just do a simple command like:

```
$ sudo ls
```

You should be prompted for your user password and then the command should execute. If instead, you get an error message you need to execute the following procedure.

Launch a root shell by typing **su** and then giving the **root** password, not your user password.

On all recent **Linux** distributions you should navigate to the `/etc/sudoers.d` subdirectory and create a file, usually with the name of the user to whom root wishes to grant **sudo** access. However, this convention is not actually necessary as **sudo** will scan all files in this directory as needed. The file can simply contain:

```
student ALL=(ALL)  ALL
```

if the user is `student`.

An older practice (which certainly still works) is to add such a line at the end of the file `/etc/sudoers`. It is best to do so using the **visudo** program, which is careful about making sure you use the right syntax in your edit.

You probably also need to set proper permissions on the file by typing:

```
$ chmod 440 /etc/sudoers.d/student
```

(Note some **Linux** distributions may require `400` instead of `440` for the permissions.)

After you have done these steps, exit the root shell by typing `exit` and then try to do `sudo ls` again.

There are many other ways an administrator can configure **sudo**, including specifying only certain permissions for certain users, limiting searched paths etc. The `/etc/sudoers` file is very well self-documented.

However, there is one more setting we highly recommend you do, even if your system already has **sudo** configured. Most distributions establish a different path for finding executables for normal users as compared to root users. In particular the directories `/sbin` and `/usr/sbin` are not searched, since **sudo** inherits the `PATH` of the user, not the full root user.

Thus, in this course we would have to be constantly reminding you of the full path to many system administration utilities; any enhancement to security is probably not worth the extra typing and figuring out which directories these programs are in. Consequently, we suggest you add the following line to the `.bashrc` file in your home directory:

```
PATH=$PATH:/usr/sbin:/sbin
```

If you log out and then log in again (you don't have to reboot) this will be fully effective.

Chapter 2

Linux Networking Concepts and Review

2.1 Labs

Exercise 2.1: Verifying vsftp and ftp Installation

Verify that the **vsftp** server and **ftp** client packages are installed prior to starting this lab.

The server package name is usually **vsftpd** and either **ftp** or **tnftp** client packages are consistent with the solutions.

- On **OpenSUSE** systems use **zypper**
- On **Ubuntu** systems use **apt-get**
- On **CentOS** systems use **yum**

Solution 2.1

- **OpenSUSE**

```
# zypper install vsftpd tnftp
```

- **Ubuntu**

```
# apt-get install vsftpd ftp
```

- **CentOS**

```
# yum install vsftpd ftp
```

Exercise 2.2: Starting a system service manually

The **vsftpd** service may be running, please stop it if required.

```
# systemctl stop vsftpd.service
```

Start the **vsftpd** daemon manually, and verify it is running.

Solution 2.2

1. Start the daemon manually
   - On **CentOS**:
     ```
     # /usr/sbin/vsftpd /etc/vsftpd/vsftpd.conf &
     ```
   - On **Ubuntu** or **OpenSUSE**:
     ```
     # /usr/sbin/vsftpd /etc/vsftpd.conf &
     ```
2. Verify it is running
   ```
   # ps -ef | grep vsftpd
   ```
3. Use the service
   ```
   $ ftp localhost
   ```
4. Stop the daemon
   ```
   # killall vsftpd
   ```

Exercise 2.3: Optional Starting a system service with the SYSV init script if they exist on your system

Start the **vsftpd** daemon using the SYSV init script, and verify that it is running.
Systems using **systemd** or **upstart** may not have SYSV scripts.

Solution 2.3

1. Start the daemon using the SYSV init script.
   ```
   # /etc/init.d/vsftpd start
   ```
2. Verify it is running
   ```
   # ps -ef | grep vsftpd
   ```
3. Use the service
   ```
   $ ftp localhost
   ```
4. Stop the daemon
   ```
   # /etc/init.d/vsftpd stop
   ```

Exercise 2.4: Starting a system service with systemd

Start the **vsftpd** daemon using **systemctl**, and verify that it is running

Solution 2.4

1. Start the daemon
   ```
   # systemctl start  vsftpd.service
   ```

2. Verify it is running

```
# ps -ef | grep vsftpd
or
# systemctl status vsftpd.service
```

3. Use the service

```
$ ftp localhost
```

4. Stop the daemon

```
# systemctl stop vsftpd.service
```

Exercise 2.5: Enable a system service using the systemctl command

Enable the **vsftpd** daemon using the distribution appropriate command.
Verify **systemd** will automatically start the service on reboot.
(Hint: the service needs to be **enabled**.)

Solution 2.5

- Enable the **vsftpd** daemon using the distribution appropriate command.

```
# systemctl enable vsftpd.service
```

- Verify the **vsftpd** service is enabled.

```
# systemctl status vsftpd.service
```

Exercise 2.6: Create and customize a systemd service

This exercise is going to explore the various configuration directories for a **systemd** service. The application **stress** will be used, install the package **stress** with the appropriate package installer.

The contents of the **stress** package does not include a **systemd** unit configuration so one must be created. The package installed on the test system has the binary for **stress** as `/usr/bin/stress`. Create a **systemd** vendor unit file as `/usr/lib/systemd/system/foo.service`

```
$ cat /usr/lib/systemd/system/foo.service

[Unit]
Description=Example service unit to run stress
[Service]
ExecStart=/usr/bin/stress --cpu 4 --io 4 --vm 2 --vm-bytes 1G
[Install]
WantedBy=multi-user.target
```

Once the unit file is created **systemd** will be able to start and stop the service. Use the **top** command to verify that **stress** is working. The following commands may be useful:

```
# systemctl start foo
# systemctl status foo -l
# systemd-delta
# systemctl stop foo
```

Since the /usr/lib/systemd/system/foo.service may be altered by the vendor at update time, create a unit file in /etc/systemd/system/foo.service for the **stress** service and change the parameters slightly so it is easy to identify which unit file is being used. It is normal to copy the vendor unit file into the /etc/systemd/system/ directory and make appropriate customizations.

```
$ cat /etc/systemd/system/foo.service

[Unit]
Description=Example service unit to run stress
[Service]
ExecStart=/usr/bin/stress --cpu 2 --io 2 --vm 4 --vm-bytes 1G
[Install]
WantedBy=multi-user.target
```

Start or restart the service and examine the differences in the following command output.

```
# systemctl status foo -l
# systemd-delta
```

Often times it is desirable to add or change features by program or script control, the drop-in files are convenient for this. One item of caution, if one is changing a previously defined function (like ExecStart) it must be undefined first then added back in. Create a drop-in directory and file for out **stress** service and verify the changes are active. Here is our example file:

```
$ cat /etc/systemd/system/foo.service.d/00-foo.conf

[Service]
ExecStart=
ExecStart=/usr/bin/stress --cpu 1 --vm 1 --io 1 --vm-bytes 1G
```

Start or restart the service and examine the differences in the following command output.

```
# systemctl status foo -l
# systemd-delta
```

With **systemd** additional features and capabilities can be easily added. As an example **cgroups** controls can be added to out service. Here is an example of adding a **systemd slice** to the example service and adding a resource limit to that slice. The slice is then attached to the service drop-in file. First setup a <service>.slice unit file:

```
$ cat /etc/systemd/system/foo.slice

[Unit]
Description=stress slice
[Slice]
CPUQuota=30%
```

Then connect our service to the **slice**. Add the following to the bottom of the unit file in /etc/systemd/system/foo.service.d/00-foo.conf

```
Slice=foo.slice
```

Restart the services and examine the differences with:

```
# systemctl daemon-reload
# systemctl status foo -l
# systemd-delta
# top
```


Bonus step: In our example there are no unique values in the `/etc/systemd/system/foo.service` file so in this example it is redundant. We can get rid of the extra file.

```
# mv /etc/systemd/system/foo.service /root/
# systemctl daemon-reload
# systemctl restart foo
# systemctl status foo
```

Consult the man pages **systemd.resource-control(5), systemd.service(5)** and other systemd man pages for additional information.

Chapter 3

Network Configuration

3.1 Labs

Exercise 3.1: Explore and Record the Existing Network Configuration

The default configuration for most of the distributions use **DHCP** to configure the interfaces discovered the first time the system is started. This lab exercise is to record the initial network values.

Solution 3.1

- Record the IP address, netmask or prefix and the network device.

```
# ip address show
```

- Record the default route

```
# ip route show
```

- Record the DNS search list and name server records

```
# cat /etc/resolv.conf
```

Exercise 3.2: Create a boot-time configuration of your network interface

Using the values recorded in the previous step, create the appropriate network configuration files. The following files have test data, be sure to use your actual values.

Make a backup copy in /var/tmp of any file before editing.

Reboot and verify the network connections function.

Solution 3.2

- On an **Ubuntu** system, edit the file `/etc/network/interfaces` and add or modify the configuration like below, using your discovered values:

```
# interfaces(5) file used by ifup(8) and ifdown(8)
auto lo
iface lo inet loopback

auto enp0s3
iface enp0s3 inet static
        address 10.0.2.15
        netmask 255.255.255.0
        gateway 10.0.2.2

#place dns information after interface stanzas
dns-nameservers 8.8.8.8
```

- On a **CentOS** system, edit the file `/etc/sysconfig/network-scripts/ifcfg-<adaptername>` file and ensure it has the following contents:

```
DEVICE=<adapter-name>
TYPE=ethernet
BOOTPROTO=none
IPADDR=10.0.2.15
PREFIX=24
GATEWAY=10.0.2.2
DNS1=8.8.8.8
NAME="LFSstatic"
ONBOOT=yes
```

- On a **SUSE** system You have to turn off **NetworkManager** first

  1. As the 'root' user run the command `yast lan`
  2. You will see a warning telling you that network manager is managing the network settings, click `OK`
  3. Under `Global Options` -> `Network Setup Method`, select `Wicked Service`
  4. Click `OK`
  5. Add the following settings in the file `/etc/sysconfig/network/ifcfg-eth0`

     ```
     NAME="LFSstatic"
     DEVICE=eth0
     BOOTPROTO="static"

     IPADDR=10.0.2.15/24

     STARTMODE="auto"
     USERCONTROL="no"
     ```

  6. Add the following to the file `/etc/sysconfig/network/config`

     ```
     NETCONFIG_DNS_STATIC_SERVERS=8.8.8.8
     ```

  7. Add the following to the file `/etc/sysconfig/network/ifroute-eth0`

     ```
     #Destination         Gateway          Netmask    Device
     default               10.0.2.2  0.0.0.0    eth0
     ```

- Once you've made the configuration changes restart the networking services using the distribution's method

Exercise 3.3: Changing Network Configuration and aliases

Create a runtime configuration change

Create a new network alias and if possible test it.

Set up the address `10.200.45.100/255.255.255.0`

If you have a second machine, set up the address `10.200.45.110/255.255.255.0`

Solution 3.3

Your machine:

```
# ip addr add 10.200.45.100/24 dev eth0
```

Another machine:

```
# ip addr add 10.200.45.110/24 dev eth0
```

Test the link from your machine:

```
$ ping 10.200.45.110
```

Test the link from the other machine:

```
$ ping 10.200.45.100
```

Exercise 3.4: Restore the DHCP configuration

Remove the files and changes to restore the **DHCP** configuration of your systems.

Solution 3.4

- On **Ubuntu** systems edit the changes out of the `/etc/network/interfaces` file.
- On **CentOS** systems remove the `/etc/sysconfig/network-scripts/ifcfg-<interfacename>` file.
- On **OpenSUSE** systems edit the changes out of the `/etc/network/config`, `/etc/network/ifcfg-<interface>` and `/etc/ifroute-<interface>` files.

Chapter 4

Network Troubleshooting and Monitoring

4.1 Labs

This lab exercise is designed to be completed on a single system. If desirable, you may use a second system to act as client to enhance the lab experience.

Exercise 4.1: Create an intermittent network issue and prove what is broken

Use this **netem** command to introduce a network problem on the server (random packet drops).

```
# tc qdisc add dev lo root netem loss random 40
```

Note: for **CentOS6** use:

```
# tc qdisc add dev lo root netem corrupt 30%
```

Prove the problem is random packet drops.

Solution 4.1

1. Use the **ping** command to verify packets are dropping.

   ```
   $ ping localhost
   ```

   You should see dropped packets.

2. While the **ping** is running, use **tcpdump** on another terminal session to see the packets sent and received.

   ```
   # tcpdump -i lo  proto ICMP
   ```

   You should see all the requests being sent, but a smaller number of responses.

3. Clean up the **tc** command to avoid issues in future labs

   ```
   # tc qdisc del dev lo root
   ```

Exercise 4.2: Prove a service is listening only on `localhost`

Prove the default **SMTP** service is only running on localhost.

NOTE: You may have to start or enable the SMTP service, and install the **telnet** client for this lab.

NOTE: **Ubuntu** systems may require reconfiguration of **postfix** before it will start. To reconfigure:

```
$ sudo dpkg-reconfigure postfix
```

Then select configuration of **local only** and accept the defaults as presented.

Solution 4.2

- Use the **netstat** command to prove where the SMTP daemon is listening.

  ```
  # netstat -taupe | grep smtp
  ```

- Use the **telnet** command to prove that the daemon is listening on localhost only.

  1. Connect to the localhost interface.

     ```
     $ telnet localhost 25
     ```

  2. Attempt to connect from a remote interface.

     ```
     $ telnet X.Y.Z.A 25
     ```

Exercise 4.3: Block traffic to a service with TCP Wrappers and prove it is blocked

Use **TCP Wrappers** to block access to FTP daemon and prove it is blocked

Solution 4.3

NOTE: The **TCP Wrappers** option is disabled by default in **Ubuntu** and **Debian**. You must add the following line to the end of the `/etc/vsftpd.conf` file to enable this feature.

```
tcp_wrappers=yes
```

1. Start a service

   ```
   # /etc/init.d/vsftpd start
   ```

2. Check port with **telnet**

   ```
   $ telnet localhost ftp
   ```

3. Block the port by adding the following line to `/etc/hosts.deny`

   ```
   vsftpd: ALL
   ```

   NOTE: On **OpenSUSE** the version of **vsftpd** is not compiled with **TCP Wrappers** support. You may alternatively use the following **iptables** command to block the FTP traffic.

   ```
   # iptables -A INPUT -m tcp -p tcp --dport ftp -j REJECT
   ```

4. Check port with **telnet**

   ```
   $ telnet localhost ftp
   ```


You should get a connection refused message. Note: The loopback may still work, use a different adapter.

5. Remove the line from `/etc/hosts.deny` to clean up the exercise. Or if you created an **iptables** rule to block traffic, flush the rules:

```
# iptables -F
```


Chapter 5

Remote Access

5.1 Labs

Exercise 5.1: Set up SSH key-based authentication

Connect to your `localhost` machine with an SSH key.

Note: This lab assumes that root is allowed to login via a password through ssh. There is a configuration parameter that can change this behavior. The different distributions may change this parameter.The parameter may also change release to release. Check the parameter `PermitRootLogin` is set to `yes` in the `/etc/ssh/sshd_config` file. If it is not, set the parameter to `yes` and restart the sshd server.

Solution 5.1

1. Make an SSH key and add it to an SSH agent.

```
$ ssh-keygen -t rsa -f $HOME/.ssh/id-rsa
$ eval $(ssh-agent)
$ ssh-add $HOME/.ssh/id-rsa
```

2. Copy the SSH pubkey manually or use **ssh-copy-id**

```
$ ssh-copy-id student@localhost
$ ssh student@localhost
$ id
$ exit
```

Exercise 5.2: Make OpenSSH client config changes

Change the default username and create a host alias using `$HOME/.ssh/config`.

Solution 5.2

1. Edit the file `$HOME/.ssh/config` with the following contents:

```
host garply
  hostname localhost
  user root

host *
  ForwardX11 yes
```

2. Use the OpenSSH client to connect to the new alias:

```
$ ssh garply
$ hostname
$ id
$ exit
```

Exercise 5.3: Secure your OpenSSH daemon

Enable key-only root logins.

Solution 5.3

1. Edit the file `/etc/ssh/sshd_config` and make sure this line is present:

   ```
   PermitRootLogin without-password
   ```

   Restart the **sshd** daemon.

   ```
   # systemctl restart sshd.service
   ```

   NOTE: On **Ubuntu**, the ssh service is named **ssh** not **sshd**.

   Attempt to log in as root. It should fail.

   ```
   $ ssh garply
   ```

   Copy the file `/home/student/.ssh/authorized_keys` to the directory `/root/.ssh/` and make sure it is owned by the root user and group.

   ```
   # cat /home/student/.ssh/authorized_keys  >> /root/.ssh/authorized_keys
   # chown root.root /root/.ssh/authorized_keys
   # chmod 640 /root/.ssh/authorized_keys
   ```

   Log in to the host `garply` again, to prove your ssh-key login works.

   ```
   $ ssh garply
   $ id
   $ hostname
   $ exit
   ```

Exercise 5.4: Launch a remote X11 application locally

Launch the xeyes program on a remote system while displaying it locally.

NOTE: You may have to install the `xeyes` program.

Solution 5.4

1. Connect to a remote server and tunnel X11.

   ```
   $ ssh -X student@server xeyes
   ```

NOTE: If you are using **OpenSUSE** without IPv6 support you need to add/modify the following line to the file `/etc/ssh/sshd_config`:

```
AddressFamily inet
```

Exercise 5.5: Parallel ssh command execution

Configure and test the **pssh** command on the local adapters on your system.

The pssh or parallel-ssh command will send commands to many machines controlled by a text file as to what machines are used. **pssh** works best with StrictHostKeyChecking=no or previously added fingerprints to the `~/.ssh/knownhosts` file. The **pssh** commands are most secure with ssh key copied into the targets authorized_keys file.

Note: Some distros use the names like `pssh`, others use `parallel-ssh` to avoid conflicts with other software use the appropriate package management command to verify installation and the names being used.

Solution 5.5

1. Install or verify **pssh** is installed

   - **Ubuntu:**

     ```
     $ sudo apt-get update
     $ sudo apt-get install pssh
     ```

     Verify the program names

     ```
     $ dpkg-query -L pssh | grep bin
     ```

   - **CentOS:**

     ```
     $ sudo yum install pssh
     ```

     Verify the program names

     ```
     $ sudo rpm -ql  pssh | grep bin
     ```

   - **OpenSUSE:**

     ```
     $ sudo zypper install  pssh
     ```

     Verify the program names

     ```
     $ sudo rpm -ql  pssh | grep bin
     ```

2. Setup ssh keys and fingerprints If not already done, create a key pair on the local machine

   ```
   $ ssh-keygen
   ```

   Copy the key to the remote and save the fingerprint

   ```
   $ ssh-copy-id localhost
   ```

3. Test the paswordless connection, if you are prompted for anything fix it now

   ```
   $ ssh localhost
   ```

   Repeat for all the local interfaces or some remotes

   ```
   $ ssh-copy-id 127.0.0.1
   $ ssh-copy-id 172.16.104.135
   ```

4. Create a ip-list file for pssh

   ```
   $ echo "127.0.0.1"  > ~/ip-list
   $ echo "172.16.104.135" >> ~/ip-list
   $ echo "localhost" >> ~/ip-list
   ```


5. Now try some commands with parallel-ssh to the local machine

```
$ parallel-ssh -i -h ~/ip-list date
$ parallel-ssh -i -h ~/ip-list sudo timedatectl
$ parallel-ssh -i -h ~/ip-list sudo hostnamectl
```

Exercise 5.6: Start a VNC server

Install VNC server.

Use the **vncserver** command to create a VNC session.

Solution 5.6

1. Ensure VNC server is installed.

   - **CentOS**

```
# yum install tigervnc-server tigervnc
```

   - **Ubuntu**

```
# apt-get install tightvncserver xtightvncviewer
```

   - **OpenSUSE**

```
# zypper install tigervnc
```

2. Start the server.

```
$ vncserver
```

3. Test the server with **vncviewer**.

```
$ vncviewer localhost:1
```

   Note: For Ubuntu users, the default `$HOME/.vnc/xstartup` file may contain references to program options that are not available. This is copy of a **xstartup** file that can be used if the vnc client only displays a gray screen. This example uses the `mwm` window manager, you may have to install the `mwm` package.

```
#!/bin/sh

xrdb $HOME/.Xresources
xsetroot -solid grey
x-terminal-emulator -geometry 80x24+10+10  &
#x-window-manager &
# Fix to make GNOME work
mwm &
export XKL_XMODMAP_DISABLE=1
/etc/X11/Xsession
```

Exercise 5.7: Tunnel VNC over SSH

Use the SSH tunneling feature of **vncviewer**.

Solution 5.7

1. Connect to a VNC server over SSH.

```
$ vncviewer -via student@hostname localhost:1
```

2. Kill your VNC server.

```
$ vncserver -kill :1
```


Chapter 6

Domain Name Service

6.1 Labs

Exercise 6.1: Configure Caching DNS

NOTE: Before starting this lab, make sure your system time is correct.

Install the BIND nameserver daemon.

Solution 6.1

1. Install the **bind** package.
   - For **Ubuntu**
     ```
     # apt-get install bind9
     ```
   - For **CentOS**
     ```
     # yum install bind
     ```
   - For **OpenSUSE**
     ```
     # zypper install bind
     ```
2. Change the configuration to allow remote access.
   - For **CentOS** and **OpenSUSE**:
     Edit the file `/etc/named.conf` and add or edit the file to contain this line inside the `options` block:
     ```
     listen-on port 53 { any; };
     allow-query     { any; };
     ```
   - For **OpenSUSE** only:
     Create an empty include file which is expected
     ```
     # touch /etc/named.conf.include
     ```

- For **Ubuntu**:
  Edit the file `/etc/bind/named.conf.options` and add or edit the file to contain these lines inside the `options` block:

```
listen-on port 53 { any; };
allow-query     { any; };
recursion yes;
```

3. Start the daemon

   - For **CentOS7** and **OpenSUSE Tumbleweed**:

```
# systemctl restart named.service
```

   - For **Ubuntu 15.04**:

```
# systemctl restart bind9.service
```

4. Test the recursive query against your nameserver.

```
$ dig @localhost google.com
```

   You should see a proper authoritative answer.

Exercise 6.2: Create an authoritative forward zone for the `example.com` domain with the following settings

- 30 second TTL
- `www.example.com` has the address `192.168.111.45` and the IPv6 address `fe80::22c9:d0ff:1ecd:c0ef`
- `foo.example.com` has the address `192.168.121.11`
- `bar.example.com` has a CNAME pointing to `www.example.com`
- `host1.example.com` through `host100.example.com` have the addresses `10.20.45.1` through `10.20.45.100`

Solution 6.2

1. Create an entry in the `named.conf` file for your new zone.

   - For **OpenSUSE** or **CentOS**:
     Edit the file `/etc/named.conf` Add a stanza like this:

```
zone "example.com." IN {
  type master;
  file "example.com.zone";
};
```

   - For **Ubuntu**:
     Edit the file `/etc/bind/named.conf.local` Add a stanza like this:

```
zone "example.com." IN {
  type master;
   file "/etc/bind/example.com.zone";
};
```

2. Create a new zone file for the `example.com` domain.

   - For **CentOS** put your zone files in the directory `/var/named/`.
   - For **OpenSUSE** put your zone files in the directory `/var/lib/named/`.

- For **Ubuntu** put your zone files in the directory `/etc/bind/`.

```
$TTL 30
@ IN SOA  localhost. admin.example.com. (
2012092901 ; serial YYYYMMDDRR format
3H         ; refresh
1H         ; retry
2H         ; expire
1M)        ; neg ttl
                IN NS localhost.;
www.example.com.  IN A 192.168.111.45
www.example.com.  IN AAAA fe80::22c9:d0ff:1ecd:c0ef
foo.example.com.  IN A 192.168.121.11
bar.example.com.  IN CNAME www.example.com.

;generate one hundred entries host1 thru host100
$GENERATE 1-100 host$.example.com.  IN A 10.20.45.$
```

3. Test your configuration with **named-checkzone** or **named-checkconf -z**
4. Restart the **named** daemon

```
# systemctl restart named
```

NOTE: the command for **Ubuntu** is:

```
# systemctl restart bind9.service
```

5. Test your new DNS entries.

```
$ dig @localhost -t A www.example.com
$ dig @localhost -t AAAA www.example.com
$ dig @localhost -t A foo.example.com
$ dig @localhost -t CNAME bar.example.com
$ dig @localhost -t A host7.example.com
$ dig @localhost -t A host37.example.com
```

Exercise 6.3: Create a reverse DNS zone for the `10.20.45.0/255.255.255.0` network listed above

Create an authoritative zone for the `45.20.10.in-addr.arpa` domain.

Solution 6.3

1. Create an entry in the `named.conf` file for your new zone.
   - For **OpenSUSE** or **CentOS** Edit the file `/etc/named.conf`.
     Add a stanza like this:

     ```
     zone "45.20.10.in-addr.arpa." IN {
       type master;
       file "45.20.10.in-addr.arpa.zone";
     };
     ```

   - For **Ubuntu** Edit the file `/etc/bind/named.conf.local`
     Add a stanza like this:

     ```
     zone "45.20.10.in-addr.arpa." IN {
       type master;
       file "/etc/bind/45.20.10.in-addr.arpa.zone";
     };
     ```

2. Create a new zone file for the ”45.20.10.in-addr.arpa” domain.

   - For **CentOS** put your zone files in the directory `/var/named/`
   - For **OpenSUSE** put your zone files in the directory `/var/lib/named/`
   - For **Ubuntu** put your zone files in the directory `/etc/bind/`

```
$TTL 30
@ IN SOA  localhost. admin.example.com. (
2012092901 ; serial YYYYMMDDRR format
3H         ; refresh
1H         ; retry
2H         ; expire
1M)        ; neg ttl
@ IN NS localhost.;
;generate 1-254
$GENERATE 1-254 $  IN PTR host$.example.com.
```

3. Test your configuration with **named-checkzone** or **named-checkconf -z**

4. Reload the **named** daemon.

```
# rndc reload
```

5. Test your new DNS entries.

```
$ host 10.20.45.7 localhost
$ host 10.20.45.37 localhost
$ host 10.20.45.73 localhost
```

Chapter 7

HTTP Servers

7.1 Labs

Exercise 7.1: Install Apache and create a simple `index.html` file to serve

Include text to indicate this is the default server.

Solution 7.1

1. Make sure **Apache** is installed:
   - On **CentOS**:
     ```
     # yum install httpd mod_ssl
     ```
   - On **OpenSUSE**:
     ```
     # zypper install apache2
     ```
   - On **Ubuntu**:
     ```
     # apt-get install apache2
     ```
2. Create an **index.html** file to serve with **Apache** in the default **DocumentRoot**.
   The contents of the **index.html** file should be:

   ```
   <html>
   <head>
     <title>This is my file</title>
   </head>
   <body>
     <h1>This is my default file</h1>
   </body>
   </html>
   ```

   The default `DocumentRoot` directories are:
   - On **CentOS, Ubuntu**:
     ```
     /var/www/html/
     ```

- On **OpenSUSE**:

  ```
  /srv/www/htdocs/
  ```

3. Make sure **Apache** is enabled and started:

   - On **systemd** distributions:

     ```
     # systemctl enable httpd
     # systemctl start httpd
     or
     # systemctl enable apache2
     # systemctl start apache2
     ```

4. Verify the page you created is visible using a web browser.

   ```
   $ firefox http://<YOUR_IP_ADDRESS>/index.html
   or
   $ w3m -dump http://<YOUR_IP_ADDRESS>/index.html
   ```

Exercise 7.2: Create a new virtual network interface and serve a different document root from the new interface

NOTE: The original html document should also be accessible from the original IP address.

1. Create an IP alias in the network `192.168.153.0/24`.
2. Serve a file indicating this is an IP based virtual machine. The file should be `/ipvhost/index.html` and **only** available on the newly defined IP address:

Solution 7.2

1. Create a temporary IP alias for your main Ethernet address:

   ```
   # ip addr add 192.168.153.X/24 dev eth0
   ```

   Where `X` is a number no one else in the same LAN is using. Add this new address to `/etc/hosts` with the host name of `ipvhost.example.com` for ease of use later.

2. Create a new directory `/ipvhost/`

   ```
   # mkdir /ipvhost/
   ```

3. Create an `/ipvhost/index.html` file.

   ```
   # vi /ipvhost/index.html
   ```

   The file should contain following:

   ```
   <html>
   <head>
     <title>This is the IP vhost</title>
   </head>
   <body>
     <h1>This is my IP vhost</h1>
   </body>
   </html>
   ```

4. Verify that **SELinux** permissions (if enabled) are correct.

   ```
   # chcon -R --reference=<YOUR-DOCUMENT-ROOT>  /ipvhost/
   ```


5. Create a new IP based virtual host definition. Add this stanza to the suggested file as listed below:

```
<VirtualHost 192.168.153.X:80>
    DocumentRoot /ipvhost/
    ServerName ipvhost.example.com
    <Directory /ipvhost/>
      Options Indexes FollowSymLinks
      AllowOverride None
      Require all granted
    </Directory>
</VirtualHost>
```

   - On **CentOS** use the file
     `/etc/httpd/conf.d/ipvhost.conf`
   - On **OpenSUSE** use the file
     `/etc/apache2/vhosts.d/ipvhost.conf`
   - On **Ubuntu** use the file
     `/etc/apache2/sites-enabled/ipvhost.conf`

6. Restart apache

```
# systemctl restart httpd
```

   NOTE: On **Ubuntu** and **OpenSUSE** the service name is `apache2`.

7. Test your new IP vhost as well as the original host.

Exercise 7.3: Create a name-based virtual host

- Create a new host name by adding the original IP address of the server to `/etc/hosts` with the name `namevhost.example.com`.
- Ensure the original web server host still serves traffic as the default vhost.
- Serve this html file on **only** the newly defined name vhost:

```
<html>
<head>
  <title>This is the namevhost</title>
</head>
<body>
  <h1>This is namevhost</h1>
</body>
</html>
```

Solution 7.3

1. Create a new name based virtual host definition. Create a new config file with the following contents, replacing the string **DOCUMENTROOT** with the proper DocumentRoot for your system.
   - On **CentOS, Ubuntu**:
     `/var/www/html/`
   - On **OpenSUSE**:
     `/srv/www/htdocs/`
   -

```
<VirtualHost *:80>
    DocumentRoot <DOCUMENTROOT>
    ServerName _default_
</VirtualHost>
<VirtualHost *:80>
    DocumentRoot /namevhost/
    ServerName namevhost.example.com
    <Directory /namevhost/>
      Options Indexes FollowSymLinks
      AllowOverride None
      Require all granted
    </Directory>
</VirtualHost>
```

- On **CentOS** use the file:
  `/etc/httpd/conf.d/namevhost.conf`
- On **OpenSUSE** use the file:
  `/etc/apache2/vhosts.d/namevhost.conf`
- On **Ubuntu** use the file:
  `/etc/apache2/sites-enabled/namevhost.conf`

2. Create the new document root folder, and create the `index.html` file:

```
# mkdir /namevhost/
# vi /namevhost/index.html
```

3. Verify that **SELinux** permissions (if enabled) are correct.

```
# chcon -R --reference=<YOUR-DOCUMENT-ROOT> /namevhost
```

4. Restart apache

```
# systemctl restart httpd
```

   NOTE: On **Ubuntu** and **OpenSUSE** the service name is `apache2`.

5. Test your new vhost as well as the original vhost.

Exercise 7.4: Create password protected sub directory

- Create the directory `secure` in the default document root.
- Require the user `bob` enter the password `heyman!` to access this directory.

Solution 7.4

1. Create the new secure folder
   - On **CentOS** and **Ubuntu** and later:
     `/var/www/html/secure/`
   - On **OpenSUSE**:
     `/srv/www/htdocs/secure/`
2. Create the following stanza to password protect the directory:

```
<Location /secure/>
  AuthType Basic
  AuthName "Restricted Area"
  AuthUserFile secure.users
  Require valid-user
</Location>
```

- On **CentOS** use the file:
  `/etc/httpd/conf.d/secure-dir.conf`
- On **OpenSUSE** use the file:
  `/etc/apache2/vhosts.d/secure-dir.conf`
- On **Ubuntu** use the file:
  `/etc/apache2/sites-enabled/secure-dir.conf`

3. Create a password file and an entry for the user `bob` in the appropriate directory:

   ```
   # htpasswd -c $FILENAME bob
   ```

   NOTE: On **OpenSUSE** the command name is **htpasswd2**.
   You may have to install **apache2-utils** if htpasswd does not exist.

   - On **CentOS** use the file:
     `/etc/httpd/secure.users`
   - On **OpenSUSE** use the file:
     `/srv/www/secure.users`

     ıtem On **Ubuntu** use the file:
     `/etc/apache2/secure.users`

4. Restart apache

   ```
   # systemctl restart  httpd
   ```

   NOTE: On **Ubuntu** and **OpenSUSE** the service name is **apache2**.

5. Verify that the directory is password protected and that `bob` is allowed to log in.

Exercise 7.5: Create and test a self-signed SSL certificate

Use the following information to create a self-signed certificate.

- Private-key pass phrase: `this is a long passphrase`
- Country Name: `US`
- State Name: `Awesome`
- Locality Name: `Awesometown`
- Organization Name: `Example Incorporated`
- Organizational Unit Name: `IT`
- Common Name: `ipvhost.example.com` where X is a unique number to your classroom or lab.
- Email Address: `admin@example.com` where X is a unique number to your classroom or lab.

Solution 7.5

1. Backup the original private key, if one exists.
   - On **CentOS**:
     ```
     # mv /etc/pki/tls/private/localhost.key /etc/pki/tls/private/localhost.key.orig
     ```
   - On **Ubuntu**:
     ```
     # mv /etc/ssl/private/ssl-cert-snakeoil.key \
          /etc/ssl/private/ssl-cert-snakeoil.key.orig
     ```
   - On **OpenSUSE**: There is no key by default so nothing needs to be backed up.
2. Create a new private key
   - On **CentOS**:
     ```
     # /usr/bin/openssl genrsa -aes128 2048 > /etc/pki/tls/private/localhost.key
     ```
   - On **OpenSUSE**:
     ```
     # /usr/bin/openssl genrsa -aes128 2048 > /etc/apache2/ssl.key/server.key
     ```
   - On **Ubuntu**:
     ```
     # /usr/bin/openssl genrsa -aes128 2048 > /etc/ssl/private/server.key
     ```
3. Create a new self-signed SSL certificate
   - On **CentOS**:
     ```
     # /usr/bin/openssl req -utf8 -new -key /etc/pki/tls/private/localhost.key -x509\
                        -days 365 -out /etc/pki/tls/certs/localhost.crt -set_serial 0
     ```
   - On **OpenSUSE**:
     ```
     # /usr/bin/openssl req -utf8 -new -key /etc/apache2/ssl.key/server.key -x509 \
                        -days 365 -out /etc/apache2/ssl.crt/server.crt -set_serial 0
     ```
   - On **Ubuntu**:
     ```
     # /usr/bin/openssl req -utf8 -new -key /etc/ssl/private/server.key -x509 \
                        -days 365 -out /etc/ssl/certs/server.crt -set_serial 0
     ```
4. Update the **Apache** configuration (if needed)
   - On **Ubuntu**: Enable SSL vhost
     ```
     # ln -s /etc/apache2/sites-available/default-ssl.conf /etc/apache2/sites-enabled/
     ```
     Enable SSL module and configuration
     ```
     # ln -s /etc/apache2/mods-available/ssl.conf /etc/apache2/mods-enabled/
     # ln -s /etc/apache2/mods-available/ssl.load /etc/apache2/mods-enabled/
     ```
     Edit the file `/etc/apache2/sites-enabled/default-ssl.conf` and modify the paths for the key and crt files so they look like this:
     ```
     SSLCertificateFile /etc/ssl/certs/server.crt
     SSLCertificateKeyFile /etc/ssl/private/server.key
     ```
     Note: You may have to comment out the directives **SSLSessionCache** and **SSLSessionCacheTimeout** from the **/etc/apache2/mods-enabled/ssl.conf** file.
   - On **OpenSUSE**: Enable SSL vhost
     ```
     # cp /etc/apache2/vhosts.d/vhost-ssl.template /etc/apache2/vhosts.d/vhost-ssl.conf
     ```
     Enable the SSL server module, edit the file `/etc/sysconfig/apache2` and add the string "SSL" to the variable `APACHE_SERVER_FLAGS` so it looks like this:
     ```
     APACHE_SERVER_FLAGS="SSL"
     ```


- On **CentOS**:
  There are no configuration changes needed.

5. Restart **Apache** and test your new certificate. You may have to add `ipvhost.example.com` to your `/etc/hosts` file.

   - On **CentOS**:
     ```
     # systemctl restart  httpd
     ```
   - On **Ubuntu** or **OpenSUSE**:
     ```
     # systemctl restart apache2
     ```

Exercise 7.6: Create a Certificate Signing Request

Use the same settings in the last exercise to generate a CSR.

Solution 7.6

1. Create a new private key

   - On **CentOS**:
     ```
     # /usr/bin/openssl genrsa -aes128 2048 > /etc/pki/tls/private/ipvhost.example.com.key
     ```
   - On **OpenSUSE**:
     ```
     # /usr/bin/openssl genrsa -aes128 2048 > /etc/apache2/ssl.key/server.key
     ```
   - On **Ubuntu**:
     ```
     # /usr/bin/openssl genrsa -aes128 2048 > /etc/ssl/private/server.key
     ```

2. Create a new CSR.

   - On **CentOS**:
     ```
     # /usr/bin/openssl req -utf8 -new -key \
                      -key /etc/pki/tls/private/ipvhost.example.com.key \
                      -out /etc/pki/tls/certs/ipvhost.example.com.csr
     ```
   - On **OpenSUSE**:
     ```
     # /usr/bin/openssl req -utf8 -new \
                       -key /etc/apache2/ssl.key/server.key \
                       -out /etc/apache2/ssl.csr/server.csr
     ```
   - On **Ubuntu**:
     ```
     # /usr/bin/openssl req -utf8 -new \
                       -key /etc/ssl/private/server.key \
                       -out /etc/ssl/server.csr
     ```

   You'll be asked for a challenge password. Make sure you remember it.

3. You must then send off this CSR to be signed by a Certificate Authority.

Chapter 8

Advanced HTTP Servers

8.1 Labs

Exercise 8.1: Create a new cgi script-enabled directory `/new-cgi/` served at the URI `/scripts/`.

Create the script `/new-cgi/foo.cgi` with the following contents (you may have to create the directory `/new-cgi/`):

```
#!/bin/bash
echo -e "\n"
echo -e "Content-type: text/plain\n\n"
echo -e "File is $1\n"
```

Solution 8.1

1. Verify your script has execute permissions.

   ```
   $ chmod +x /new-cgi/foo.cgi
   ```

2. Create a configuration **include** file in the location suggested below which, enables **cgi-scripts** for the `/scripts/` URI.

   ```
   ScriptAlias /scripts/ /new-cgi/
   <Directory /new-cgi/>
           Require all granted
   </Directory>
   ```

   - On **CentOS** use the file:
     `/etc/httpd/conf.d/newscripts.conf`
   - On **Ubuntu** use the file:
     `/etc/apache2/sites-enabled/newscripts.conf`
   - On **OpenSUSE** use the file:
     `/etc/apache2/conf.d/newscripts.conf`

3. If required, enable the cgi module:
   - On **Ubuntu** and **Debian**: Enable the cgi module to be loaded.
     ```
     # ln -s /etc/apache2/mods-available/cgi.load /etc/apache2/mods-enabled/
     or
     # a2enmod cgi
     ```
4. Restart **Apache** and test your new script. http://localhost/scripts/foo.cgi?bar

Exercise 8.2: Create a rewrite rule for "pretty" CGI script URIs

NOTE: If you have done the virtualhost lab from the previous chapter you need to add these two lines inside of the `namevhost.conf` file in the `_default_` namevhost section:

```
RewriteEngine on
RewriteOptions inherit
```

Make any URIs which begins with:

```
http://localhost/foo/.*
```

Redirect transparently to the cgi script:

```
http://localhost/scripts/foo.cgi?.*
```

Solution 8.2

1. Create a configuration include file in the suggested location below which sets up the proper rewrite rules:
   ```
   RewriteEngine on
   RewriteRule ^/foo/(.*) /scripts/foo.cgi?$1 [L,PT]
   ```
   - On **CentOS** use the file
     `/etc/httpd/conf.d/rewrite.conf`
   - On **Ubuntu** use the file
     `/etc/apache2/sites-enabled/rewrite.conf`
   - On **OpenSUSE** use the file
     `/etc/apache2/conf.d/rewrite.conf`
   - **Note:** on **Ubuntu** and **Debian** systems the **rewrite** commands seem to work best in the **000-default.conf** file. If using **Ubuntu** or **Debian** systems put the following inside the virtualhost stanza of the **000-default.conf** file.
     ```
     RewriteEngine on
     RewriteOptions inherit
     RewriteRule ^/foo/(.*) /scripts/foo.cgi?$1  [L,PT]
     ```
2. If required, enable the rewrite module:
   - On **OpenSUSE**: Edit the file `/etc/sysconfig/apache2` and edit the line with the `APACHE_MODULES`, and add the value `rewrite`
   - On **Ubuntu**: Enable the rewrite module to be loaded
     ```
     # ln -s /etc/apache2/mods-available/rewrite.load /etc/apache2/mods-enabled/
     ```


3. Restart **Apache** and test your new URI.

   http://localhost/foo/bar

Exercise 8.3: Enable Mod_Status

Secure mod_status to be accessible to only the network 10.22.34.0/18, ::1 and 127.0.0.1.

Solution 8.3

1. Create a configuration include file in the suggested location which enables mod_status.

```
<Location /server-status/>
  SetHandler server-status
  Require ip 10.22.34.0/18 ::1 127.
</Location>
```

   - On **CentOS** use the file:
     /etc/httpd/conf.d/status.conf
   - On **Ubuntu**, make sure the module config file is correct instead of creating a new file.
     /etc/apache2/mods-available/status.conf
   - On **OpenSUSE** use the file:
     /etc/apache2/conf.d/status.conf

2. Confirm or create the server-status directory exists in your distributions DOCUMENTROOT.
   - On **CentOS** and **Ubuntu14** and later:
     /var/www/html/server-status/
   - On **OpenSUSE**:
     /srv/www/htdocs/server-status/
   -

3. If required, enable the **status** module:
   - On **OpenSUSE**:
     Edit the file /etc/sysconfig/apache2 and edit the line with the APACHE_MODULES, and add the value status
   - On **Ubuntu**: Enable the **status** module to be loaded

```
# ln -s /etc/apache2/mods-available/status.load /etc/apache2/mods-enabled/
# ln -s /etc/apache2/mods-available/status.conf /etc/apache2/mods-enabled/
```

4. Restart **Apache** and test your new URI.

   http://localhost/server-status/

Exercise 8.4: Enable includes under the URI /magic/index.html

Include the two files foo.html and bar.html that are located in the DOCUMENTROOT/includes directory. Ensure that files with the extension .html are processing includes, but only when needed.

Solution 8.4

1. Create the following html file for the /magic/ URI:

```
<html>
<head>
  <title>This file is a magic include file</title>
</head>
<body>
  <h1>This file is a magic include file</h1>
  <h2>Foo include below</h2>
  <!--#include virtual="/includes/foo.html" -->
  <h2>Bar include below</h2>
  <!--#include virtual="/includes/bar.html" -->
</body>
</html>
```

- On **CentOS** use the file:
  `/var/www/html/magic/index.html`
- On **OpenSUSE** use the file:
  `/srv/www/htdocs/magic/index.html`
- On **Ubuntu** use the file:
  `/var/www/magic/index.html`

2. Create the two files to be included in the main page using the content and locations suggested below.

```
this is the foo include
```

- On **CentOS** use the file:
  `/var/www/html/includes/foo.html`
- On **OpenSUSE** use the file:
  `/srv/www/htdocs/includes/foo.html`
- On **Ubuntu** use the file:
  `/var/www/magic/includes/foo.html`

```
this is the bar include
```

- On **CentOS** use the file:
  `/var/www/html/includes/bar.html`
- On **OpenSUSE** use the file:
  `/srv/www/htdocs/includes/bar.html`
- On **Ubuntu** use the file:
  `/var/www/magic/includes/bar.html`

3. Create a configuration **include** file in the suggested location listed below which enables **includes**.

```
<Location /magic/>
  Options +Includes
  XBitHack on
</Location>
```

- On **CentOS** use the file:
  `/etc/httpd/conf.d/magic.conf`
- On **Ubuntu** use the file:

/etc/apache2/sites-enabled/magic.conf

- On **OpenSUSE** use the file:
  /etc/apache2/conf.d/magic.conf

4. If required, enable the **include** module:
   - On **Ubuntu**:
     ```
     # ln -s /etc/apache2/mods-available/include.load /etc/apache2/mods-enabled/
     ```

   Ensure that your `magic/index.html` file is executable.

5. Restart **Apache** and test your new URI.

   http://localhost/magic/index.html

Chapter 9

Email Servers

9.1 Labs

Exercise 9.1: Enable the Postfix SMTP server for external access

- Ensure all hosts in your network are allowed to send email to your server.

Solution 9.1

1. Ensure **Postfix** is installed:
   - On **CentOS**:
     ```
     # yum install postfix
     ```
   - On **OpenSUSE**:
     ```
     # zypper install postfix
     ```
   - On **Ubuntu** and **Debian**:
     ```
     # apt-get install postfix
     ```
     - If asked what method for configuration type choose `Internet Site`.
     - If asked which `mail name`, set it to your current host name.
2. Enable **Postfix** to listen on all interfaces:
   ```
   # postconf -e "inet_interfaces = all"
   ```
3. Enable trusted subnets:
   ```
   # postconf -e "mynetworks_style = subnet"
   ```
4. Restart **Postfix**:
   ```
   # systemctl restart postfix
   ```

Note: Be aware the firewall may interfere with this test.

5. Test from a remote server using **telnet**
(you may need to install **telnet**):

   Note: The commands (like helo,mail,rcpt, etc) may need to be capitalized on some distributions.

```
$ telnet <IP ADDRESS> 25
helo localhost
mail from:root@localhost
rcpt to:root@localhost
data
Subject: testing telnet email

This is neato
.
quit
```

6. Verify the mail was received using the **mutt** command or the **mail** command.

   NOTE: you may have to install the `mail` command. It is part of either the `mailx` (**CentOS** or **OpenSUSE**) or `mailutils` (**Ubuntu** and **Debian**) packages.

Exercise 9.2: Enable dovecot as IMAP server

- Ensure the `student` user can log in to IMAP using the password `student`.
- Prepare for this lab by sending a couple of emails to the `student` user:

```
for i in one two three;
do
echo $i | mail -s "test $i" student@localhost;
done
```

Solution 9.2

1. Ensure **dovecot** and `mutt` are installed:
   - On **CentOS**:

     ```
     # yum install dovecot mutt
     ```

     On **OpenSUSE**:

     ```
     # zypper install dovecot20 mutt
     ```

     On **Ubuntu** and **Debian**:

     ```
     # apt-get install mutt dovecot-imapd dovecot-pop3d dovecot-core dovecot-lmtpd
     ```

2. Ensure **dovecot** is listening on all interfaces for the IMAP protocol,
   - On **CentOS** and **OpenSUSE**:
     Edit the file `/etc/dovecot/dovecot.conf` and add/modify these lines:

     ```
     protocols = imap pop3 lmtp
     listen = *
     ```

   - On **Ubuntu** and **Debian**:
     (a) Remove the configuration file for the **Sieve** protocol if it exists.

     ```
     # rm -f /usr/share/dovecot/protocols.d/managesieved.protocol
     ```

     (b) Create a new configuration for the **LMTP** protocol if required.


```
# echo 'protocols = $protocols lmtp' > \
                    /usr/share/dovecot/protocols.d/lmtp.protocol
```

(c) Edit the file `/etc/dovecot/dovecot.conf` and add/modify these lines:

```
listen = *
```

3. Ensure **dovecot** is using the proper storage location for email:

   - On **OpenSUSE** or **CentOS**: Edit the file `/etc/dovecot/conf.d/10-mail.conf` and add the line:

     ```
     mail_location = mbox:~/mail:INBOX=/var/spool/mail/%u
     ```

   - On **Ubuntu** and **Debian**:
     Review the contents of the default mail delivery configuration file `/etc/dovecot/conf.d/10-mail.conf`. No changes need to be made as the default has the proper settings already.

4. Restart **dovecot**:

   ```
   # systemctl restart dovecot
   ```

5. Test the **dovecot** server using **mutt**

   ```
   $ mutt -f imap://student@<IP_ADDRESS>/
   ```

   - NOTE: You may have to connect twice with **mutt** to verify the **imap** server is working.
   - NOTE: The server may already be set up for SSL/StartTLS with a dummy SSL certificate.
   - NOTE: There may be a permission challenge creating mail directories. Look in the mail log to confirm the create directory error. Add the group **mail** to the **student** account temporarily.

Exercise 9.3: Enforce TLS/SSL for IMAP in `dovecot`

Solution 9.3

1. Edit the configuration file and require ssl.

   - On **CentOS**: Edit the file `/etc/dovecot/conf.d/10-ssl.conf` and add or edit these line:

     ```
     ssl = required
     ```

   - On **OpenSUSE**:

     (a) Add or edit the file `/etc/dovecot/conf.d/10-ssl.conf` and make sure these lines exist:

     ```
     ssl = required
     ssl_cert = </etc/ssl/certs/dovecot.pem
     ssl_key = </etc/ssl/private/dovecot.pem
     ```

     (b) Generate a self-signed certificate for IMAP.

     ```
     # cd /usr/share/doc/packages/dovecot/
     # ./mkcert.sh
     ```

   - On **Ubuntu** and **Debian**:

     (a) Edit the file `/etc/dovecot/conf.d/10-ssl.conf` and change the lines:

     ```
     ssl = required
     ssl_cert = </etc/dovecot/dovecot.pem
     ssl_key = </etc/dovecot/private/dovecot.pem
     ```

     (b) Generate a self-signed certificate for IMAP.

```
# cd /usr/share/dovecot
# ./mkcert.sh
```

2. Restart **dovecot**: and test your **mutt** command again.

```
# systemctl restart dovecot
$ mutt -f imap://student@<IP_ADDRESS>/
```

Exercise 9.4: Enable relaying using SMTP Auth in `postfix`

- Ensure the `mynetworks_style` is set to `host`:

```
# postconf -e "mynetworks_style = host"
```

To avoid issues with incorrectly set up DNS, or enforced **ssl**, use this setting for your lab as well:

```
# postconf -e "disable_dns_lookups = yes"
# postconf -e "smtpd_tls_auth_only = no"
```

NOTE: Don't enable these settings in production. Use them only for this lab.

NOTE: We will re-enforce **SSL** authentication in the next exercise.

- Restart **Postfix** with above setting before starting the lab:

```
# systemctl restart postfix
```

Solution 9.4

1. Enable the **SASL** authentication service in **Dovecot**.
   - Edit the file `/etc/dovecot/conf.d/10-master.conf` and after the section `service auth` add or un-comment the following lines:

```
unix_listener /var/spool/postfix/private/auth {
  mode = 0666
}
```

2. Restart **Dovecot**:

```
# systemctl restart dovecot
```

3. Enable sasl authentication in **Postfix**.
   Make the following setting changes:

```
# postconf -e "smtpd_sasl_type = dovecot"
# postconf -e "smtpd_sasl_auth_enable = yes"
# postconf -e "smtpd_recipient_restrictions = \
             permit_mynetworks,              \
             permit_sasl_authenticated,      \
             reject_unauth_destination"
```

4. Configure the proper authentication path:

```
# postconf -e "smtpd_sasl_path = private/auth"
```

5. Restart **Postfix**:

```
# systemctl restart postfix
```

6. Test plain text authentication from a **remote host**.
   Notice that any system on our subnet will be allowed to relay due to **permit_mynetworks**.
   If you wish to test on a single machine eliminate the **permit_mynetworks** entry from **smtpd_recipient_restrictions** to force all relaying to authenticate.

   ```
   $ telnet <SERVER> 25
   helo localhost
   mail from:student
   rcpt to:root@<OTHER MACHINE>
   quit
   ```

   This should fail with relay access denied. Test again with authentication:
   Create the base64 encoded user and password.

   ```
   $ echo -en "\0student\0student" | base64
   ```

   Using the encrypted user and password, send the email.

   ```
   $ telnet <SERVER> 25
   helo localhost
   auth plain AHN0dWRlbnQAc3R1ZGVudA==
   mail from:student
   rcpt to:root@<OTHER MACHINE>
   data
   Subject: I sent this using SASL SMTP auth

   Cool no?
   .
   quit
   ```

Exercise 9.5: Enable StartTLS for Postfix, and force Plain-Text logins to use StartTLS

Use the following information to create a certificate.

- Private-key pass phrase: `this is a long passphrase`
- Country Name: `US`
- State Name: `Awesome`
- Locality Name: `Awesometown`
- Organization Name: `Example Incorporated`
- Organizational Unit Name: `IT`
- Common Name: `smtp.example.com`
- Email Address: `admin@smtp.example.com`

Solution 9.5

1. Create a new PEM certificate:
   - For **CentOS**:

     ```
     # cd /etc/pki/tls/certs
     # make postfix.pem
     ```

   - For other distributions:

```
# /usr/bin/openssl req -utf8 -newkey rsa:2048  -keyout /tmp/postfix.key -nodes \
                       -x509 -days 365 -out /tmp/postfix.crt -set_serial 0
# cat /tmp/postfix.key > /etc/postfix/postfix.pem
# echo "" >> /etc/postfix/postfix.pem
# cat /tmp/postfix.crt >> /etc/postfix/postfix.pem
# rm -f /tmp/postfix.crt /tmp/postfix.key
```

- Change the **Postfix** configuration to enable and enforce TLS:

```
# postconf -e "smtpd_tls_auth_only = yes"
# postconf -e "smtpd_tls_security_level = may"
# postconf -e "smtpd_tls_cert_file = /etc/postfix/postfix.pem"
# postconf -e "smtpd_tls_key_file = /etc/postfix/postfix.pem"
```

- Restart **Postfix**:

```
# systemctl restart postfix
```

- Test SMTP StartTLS:
  Note: You may have to do this twice to get the key data.
  Note: After the **starttls** command use the ”control + d” key combination.

```
$ gnutls-cli --crlf --starttls --insecure --port 25 <IP ADDRESS>
ehlo <HOSTNAME>
starttls
^d
auth plain AHNOdWRlbnQAc3R1ZGVudA==
mail from:student
rcpt to:root@<LOCAL IP ADDRESS>
data
Subject: I sent this using SASL SMTP auth protected by TLS

Cool no?
And secure!
.
quit
```

NOTE: There is no option for AUTH until after you start the TLS session.
NOTE: Relay access is still denied until after the AUTH step.

Chapter 10

File Sharing

10.1 Labs

Exercise 10.1: Use SCP to copy a folder from one location to another. Create a directory full of testing files to use for this lab:

```
$ mkdir /tmp/transfer-lab/
$ mkdir /tmp/receive/
$ for i in /tmp/transfer-lab/{a,b,c}-{1,2,3}.{txt,log,bin}
  do
    echo $i > $i
  done
```

Use **scp** to copy just the `.log` files from `/tmp/transfer-lab/`, into `/tmp/receive/` through the loopback interface.

Solution 10.1

```
$ scp /tmp/transfer-lab/*.log root@loopback:/tmp/receive
```

Exercise 10.2: Use rsync over ssh to add the `*.bin` files only to the previously created folder

Solution 10.2

Note the `.` at the end of the command.

```
$ rsync -av /tmp/transfer-lab/*.bin root@loopback:/tmp/receive/.
```

Exercise 10.3: Create a secure FTP upload site

Enable the **ftp** directory `/uploads/` for anonymous uploads.

Ensure that files uploaded, cannot be downloaded via FTP.

Solution 10.3

1. Create the upload directory with the proper permissions.
   - On **CentOS**:
     ```
     # mkdir -m 730 /var/ftp/uploads/
     # chown root.ftp /var/ftp/uploads/
     ```
   - On **Ubuntu** or **OpenSUSE**:
     ```
     # mkdir -m 730 /srv/ftp/uploads/
     # chown root.ftp /srv/ftp/uploads/
     ```
2. Edit `vsftpd.conf` and enable anonymous uploads. Add the following option:
   ```
   anon_upload_enable=yes
   anonymous_enable=yes
   ```
   NOTE: on **Ubuntu** or **OpenSUSE** you must also change the option `write_enable` to match:
   ```
   write_enable=YES
   ```
   - On **CentOS**, the path is `/etc/vsftpd/vsftpd.conf`
   - On **Ubuntu** or **OpenSUSE**, the path is `/etc/vsftpd.conf`

Exercise 10.4: Share a folder over the rsync protocol. Create a directory full of testing files to use for this lab

```
# mkdir /srv/rsync/
# for i in /srv/rsync/{a,b,c}-{1,2,3}.{txt,log,bin}
  do
    echo $i > $i
  done
```

Serve the directory `/srv/rsync/` directly via **rsync**, use the **rsync** module name of `default`.

Solution 10.4

1. Create or edit the file `/etc/rsyncd.conf` and add these contents:
   ```
   [default]
       path    = /srv/rsync
       comment = default rsync files
   ```
   NOTE: on **OpenSUSE** you must also comment out or remove the line:
   ```
   hosts allow = trusted.hosts
   ```
2. Enable the **rsync** daemon.
   - On **CentOS** or **OpenSUSE**:
     ```
     # systemctl start rsyncd
     # systemctl enable rsyncd
     ```
     NOTE: you may have to start/enable the **xinetd** daemon.
   - On **Ubuntu14**: Enable the daemon to run via the **init** script.
     ```
     # update-rc.d rsync enable 2345
     ```
     Also enable the **rsync** daemon configuration by editing the file `/etc/default/rsync` and changing the line:


```
RSYNC_ENABLE=false
```

to:

```
RSYNC_ENABLE=true
```

Start the **rsync** daemon:

```
# systemctl start rsync
```

To see the rsync modules shared:

```
$ rsync localhost::
```

Chapter 11

Advanced Networking

11.1 Labs

Exercise 11.1: Create a VLAN-trunked interface

NOTE: if you have a problem with this exercise, you may need to reboot and choose a different kernel. Create a new tagged VLAN interface with the id 7 and these settings:

- The physical interface should be the main interface (`eth0`).
- The IP address should be `192.168.X.100`, where `X` is a number unique to your classroom/lab.
- The Netmask should be `255.255.255.0`.

Solution 11.1

1. Enable VLANs:
   - On **CentOS**: edit the file `/etc/sysconfig/network` and add the following content:

     ```
     VLAN=yes
     VLAN_NAME_TYPE="DEV_PLUS_VID"
     ```

2. Create a VLAN interface configuration file:
   - On **CentOS**, edit or create the file

     `/etc/sysconfig/network-scripts/ifcfg-<INTERFACE>.7`

     with the following content:

     ```
     DEVICE=<INTERFACE>.7
     BOOTPROTO=static
     TYPE=vlan
     ONBOOT=yes
     IPADDR=192.168.X.100
     NETMASK=255.255.255.0
     PHYSDEV="<INTERFACE>"
     ```

- On **OpenSUSE**, edit or create the file

  `/etc/sysconfig/network/ifcfg-<INTERFACE>.7`

  with the following content:

```
NAME='<INTERFACE> vlan'
STARTMODE='auto'
VLAN_ID='7'
IPADDR='192.168.X.100/24'
ETHERDEVICE=<INTERFACE>
```

- On **Ubuntu**, edit the file:

  `/etc/network/interfaces`

  and add the following stanza:

```
auto <INTERFACE>.7
iface <INTERFACE>.7 inet static
        address 192.168.X.100
        netmask 255.255.255.0
        vlan-raw-device <INTERFACE>
```

  The package `vlan` may need to be installed and the kernel-module `8021q` loaded.

3. Start the newly defined interface:

```
# ifup <INTERFACE>.7
```

Exercise 11.2: Create a new static route

Creating and testing routing is always best done on two or more systems. If a second system is available whether it be a physical,virtual or container,it will be known as the **other** system. The goal of this lab is to create aliases on different networks and be able to ping the other system's address.
Create an IP alias on each system. The addresses should be on separate subnets, and require a specific route:

- On your system use the address `172.16.X.100`
- On the other system use the address `192.168.Y.100`

Where X and Y are unique to the lab/classroom.

```
# ip addr add <ADDRESS>/24 dev eth0
```

Solution 11.2

1. Create a static route configuration:

   - On **CentOS**, create or edit the file

     `/etc/sysconfig/network-scripts/route-<INTERFACE>`

     and add the following content:

     – On your system:

```
192.168.Y.0/24 via <ADDR-Y> dev <INTERFACE>
172.16.Y.0/24 via <ADDR-Y> dev <INTERFACE>
```

       NOTE: <ADDR-Y> is the original public IP address of the interface.

     – On the other system:

```
192.168.X.0/24 via <ADDR-X> dev <INTERFACE>
172.16.X.0/24 via <ADDR-X> dev <INTERFACE>
```

       NOTE: <ADDR-X> is the original public IP address of the interface.

- On **OpenSUSE**, create or edit the file:
  `/etc/sysconfig/network/ifroute-<INTERFACE>`
  and add the following content:
  - On your system:
    ```
    192.168.Y.0/24 <ADDR-Y> - <INTERFACE>
    172.16.Y.0/24 <ADDR-Y> - <INTERFACE>
    ```
  - On the other system:
    ```
    192.168.X.0/24 <ADDR-X> - <INTERFACE>
    172.16.X.0/24 <ADDR-X> - <INTERFACE>
    ```
- On **Ubuntu**, edit the file `/etc/network/interfaces` and in the stanza for `<INTERFACE>` add the following lines: (should fit on two lines)
  - On your system:
    ```
    up route add -net 192.168.Y.0/24 gw <ADDR-Y> dev <INTERFACE>
    up route add -net 172.16.Y.0/24 gw <ADDR-Y> dev <INTERFACE>
    ```
  - On the other system:
    ```
    up route add -net 192.168.X.0/24 gw <ADDR-X> dev <INTERFACE>
    up route add -net 172.16.X.0/24 gw <ADDR-X> dev <INTERFACE>
    ```

2. Restart the network:
   ```
   # systemctl restart network
   ```

3. Ping the remote address:
   - On your system:
     ```
     $ ping 192.168.Y.100 ping 172.16.Y.100
     ```
   - On the other system:
     ```
     $ ping 192.168.X.100 ping 172.16.X.100
     ```

Exercise 11.3: Configure and enable a stratum 3 NTP server. Connect your server to the NTP pool as a client

Solution 11.3

1. Ensure the NTP daemon is installed:
   - On **CentOS**:
     ```
     # yum install ntp
     ```
   - On **OpenSUSE**:
     ```
     # zypper install ntp
     ```
   - On **Ubuntu**:
     ```
     # apt-get install ntp
     ```

2. Configure the NTP server to query the NTP pool and allow for anonymous traffic.
   Edit the file `/etc/ntp.conf` and change the `server X.X.X.X` lines to match these:
   ```
   server 0.pool.ntp.org
   server 1.pool.ntp.org
   server 2.pool.ntp.org
   server 3.pool.ntp.org
   ```

3. Restart **ntp**:
   - On **CentOS**:
     ```
     # systemctl restart ntpd
     ```
   - On **OpenSUSE** or **Ubuntu**:
     ```
     # systemctl restart ntp
     ```
4. Query your new timeserver:
   ```
   $ ntpq -p
   ```
   NOTE: you may have to wait a few minutes for the timeservers to sync prior to getting any output.

Chapter 12

HTTP Caching

12.1 Labs

Exercise 12.1: Create a basic squid forward proxy

- Ensure your local network can utilize the proxy.
- Even though your RFC 1918 local network may already be in the default `squid.conf` file, explicitly set your current network as an ACL.

Solution 12.1

1. Ensure **squid** is installed:
   - On **CentOS**:
     ```
     # yum install squid
     ```
   - On **OpenSUSE**:
     ```
     # zypper install squid
     ```
   - On **Ubuntu**:
     ```
     # apt-get install squid
     ```
2. Create an ACL for your network, edit the file `/etc/squid/squid.conf` and add the following just after the line which reads:

   `# INSERT YOUR OWN RULE(S) HERE TO ALLOW ACCESS FROM YOUR CLIENTS`

   NOTE: on some **Ubuntu** systems the file may located at `/etc/squid3/squid.conf` and may be very verbose. The place in the file to insert rules is `above` the line `http_access allow localhost`.

   ```
   acl examplenetwork src <NETWORK ADDRESS>/24
   ```
3. Explicitly allow HTTP access for the newly created ACL, by adding this line below the ACL added above:

```
http_access allow examplenetwork
```

4. Test the syntax of `squid.conf`:

```
# squid -k parse
```

5. Start the Squid daemon:

   - On **CentOS** and **OpenSUSE**:

```
# systemctl  restart squid
```

   - On **Ubuntu**:

```
# systemctl restart squid3
```

6. Test the proxy:

   - Configure a web browser to use your new proxy.
   - Visit a known good URI (`http://google.com`).
   - Visit a known non-existent URI (`http://sdfa.klj.example.com`)

   NOTE: You should see a `Squid` error page when you attempt to access the non-existent URI.

Exercise 12.2: Restrict access to `http://maps.google.com/` using Squid

Solution 12.2

1. Create an ACL defining the URI to block, edit the file `/etc/squid/squid.conf` and create a new ACL above the lines you previously added:

```
acl googlemaps url_regex ^http://.*.google.com/maps/.*$
```

2. Block access to the newly created ACL `googlemaps`, edit the file `/etc/squid/squid.conf` and add the following line just above the line you added earlier, `http_access allow examplenetwork`:

```
http_access deny googlemaps
```

3. Tell **squid** to reload its configuration file:

```
# squid -k reconfigure
```

4. Test the ACL by trying to browse to `http://www.google.com/maps`

Chapter 13

Network File Systems

13.1 Labs

Exercise 13.1: Create the environment for the nfs and cifs lab

Create some directories to share and some to use as mount points.

```
# mkdir -p /home/{export,share}/{nfs,cifs}
# touch /home/export/nfs/{foo,bar,baz}.{txt,log,bin}
```

Add a group for collaboration and add **student** to the new group.

```
groupadd -g 42000 share
chown nfsnobody /home/export
chgrp share -R /home/export
chmod -R 2770 /home/export
usermod -aG share student
```

Ensure that the **NFS** server is installed:

- On **CentOS**:

  ```
  # yum install nfs-utils
  ```

- On **OpenSUSE**:

  ```
  # zypper install nfs-utils nfs-kernel-server
  ```

- On **Ubuntu**:

  ```
  # apt-get install nfs-common nfs-kernel-server
  ```

Exercise 13.2: Create NFS share:

- Share the directory `/home/export/nfs/`.

- Allow every host on your local network to read the export.
- Allow a single host on your local network to have read/write access. Initially use the loopback for ease of testing.

Solution 13.2

1. Edit the file `/etc/exports` and add the following contents:

```
/home/export/nfs  127.0.0.1/32(rw) <NETWORK ADDRESS>/24(ro)
```

2. Start or restart the NFS service:
   - On **CentOS**:

```
# systemctl restart rpcbind
# systemctl restart nfs
```

   - On **OpenSUSE**:

```
# systemctl restart rpcbind
# systemctl restart nfsserver
```

   - On **Ubuntu**:

```
# systemctl restart  nfs-kernel-server
```

3. Test the mount on two different systems.

```
# mount 127.0.0.1:/home/export/nfs /home/share/nfs
$ touch /home/share/nfs/foo
```

   If an additional systems is available on the network **NETWORK ADDRESS** mount the share on the other system. The **touch** command should only work on the host allowed read/write access, which is **127.0.0.1** in our test case.

Exercise 13.3: Create guest-access SMB share

- Share the directory `/home/export/cifs` as the share name `mainexports`.
- Use the workgroup name `LNXFND`.
- Allow read only, guest access to all hosts in the workgroup.
- Allow public access.

Solution 13.3

1. Ensure **Samba** is installed:
   - On **CentOS**:

```
# yum install samba samba-client samba-common
```

   - On **OpenSUSE**:

```
# zypper install samba samba-client
```

   - On **Ubuntu**:

```
# apt-get install samba smbclient
```

2. Backup the `smb.conf` file.

```
# mv /etc/samba/smb.conf /etc/samba/smb.conf.backup
```

3. Create a new `smb.conf` file with the following contents:

```
[global]
    workgroup = LNXFND
    server string = Myserver
    log file = /var/log/samba/log.%m
    max log size = 50
    cups options = raw

[mainexports]
    path = /home/export/cifs
    read only = yes
    guest ok = yes
    comment = Main exports share
```

4. Restart **Samba**:

   - On **CentOS** and **OpenSUSE**:

     ```
     # systemctl restart smb
     ```

   - On **Ubuntu**:

     ```
     # systemctl restart  smbd
     ```

5. Check the share is available. The password is not required, when prompted press enter to continue as **anonymous**.

   ```
   $ smbclient -L 127.0.0.1
   ```

6. From another machine or the `localhost` verify the **samba** share is working:

   ```
   $ smbclient //SERVER/mainexports
   ```

   `smbclient` will prompt for a password, you can press enter to have no password and `smbclient` will continue as anonymous user. If there is a user id and password added by `smbpasswd`, you may use those credentials. See the man page for `smbclient` for additional information.

Exercise 13.4: Create a private share for a single user

- Create and share the directory `/home/export/private/` as the share name `private` and populate it with files:

  ```
  # mkdir /home/export/private/
  # touch /home/export/private/PRIVATE_FILES_ONLY
  # chown -R student: /home/export/private
  ```

- Create a **Samba** password for the `student` account.
- Allow full access to the `student` account.

Solution 13.4

1. Edit the file `/etc/samba/smb.conf` and add this stanza to the bottom.

   ```
   [private]
       path = /home/export/private
       comment = student's private share
       read only = No
       public = No
       valid users = student
   ```

2. Add a samba password for `student`.

```
# smbpasswd -a student
```

3. From a remote system, verify the access for the student account:

```
$ smbclient -U student //SERVER/private
```

Test read and write access.

4. Create a **credential's** file with the `student` id and password for later use when adding the mounts to the fstab.

```
# echo "username=student" > /root/smbfile
# echo "password=student" >> /root/smbfile
# chmod 600 /root/smbfile
```

Exercise 13.5: Persistent Network Mounts

Using the `/etc/fstab` mount:

- The NFS share `/home/export/nfs` on the mount-point `/home/share/nfs`
- The CIFS share **mainexports** on the mount-point `/home/share/cifs`

Solution 13.5

1. Add the following to the `/etc/fstab` for the NFS mount.

```
127.0.0.1:/home/export/nfs /home/share/nfs nfs _netdev 0 0
```

2. Add the following to the `/etc/fstab` for the CIFS mount.

```
//localhost/mainexports /home/share/cifs cifs creds=/root/smbfile,_netdev 0 0
```

3. Instruct **systemd** to re-read the `/etc/fstab`

```
# systemctl daemon-reload
```

4. Mount all the Filesystem and verify they are mounted.

```
# mount -a
# df -h
```

Exercise 13.6: Convert the NFS and CIFS automatically mounted and unmounted

Convert the exported filesystems to be auto mounted by `systemd.automount` and dismount them when idle for 10 seconds.

Using the **systemd.automount** facility make previous **NFS** and **CIFS** mount automatically mount when accessed and dismount if idle for 10 seconds. Note: The idle timer will not run if a process has done a "cd" into the directory. **Solution 13.6**

1. Make the changes to the `/etc/fstab` as shown.

```
127.0.0.1:/home/export/nfs /home/share/nfs nfs x-systemd.automount,x-systemd.idletimeout=10,noauto,_netdev 0 0
//localhost/mainexports  /home/share/cifs cifs creds=/root/smbfile,x-systemd.automount,x-systemd.idletimeout=10,noauto,_netdev 0 0
```

2. Instruct **systemd** to re-read the `/etc/fstab`

```
# systemctl daemon-reload
```

3. Test the auto-mounter by displaying a file in the shared directory or by changing into the directory. Don't forget the the auto-timer does not run if the directory is busy.

```
$ df -h
$ cd /home/share/nfs
$ df -h
$ cd
```

Wait a bit and re-execute the `df` command to see that the auto-mounter dismounted the share.

Chapter 14

Introduction to Network Security

14.1 Labs

There is no lab to complete for this chapter.

Chapter 15

Firewalls

15.1 Labs

Exercise 15.1: Lock down services using tcpwrappers

- Install and test the **telnet** server.
- Using TCP wrappers, lock down all services except **sshd**.

Solution 15.1

1. Ensure that the **telnet** server is installed:
   - On **CentOS**:
     ```
     # yum install telnet-server
     ```
   - On **OpenSUSE**:
     ```
     # zypper install telnet-server
     ```
   - On **Ubuntu**:
     ```
     # apt-get install telnetd xinetd
     ```
2. Enable and test the **telnet** server: NOTE: you may have to enable the **xinetd** daemon.
   - On **CentOS** or **OpenSUSE**
     ```
     #  systemctl start telnet.socket
     #  telnet localhost
     ```
   - On **Ubuntu**:
     Create an **xinetd** configuration file for the **telnet** daemon. Edit the file `/etc/xinetd.d/telnet` and add the following contents:

```
service telnet
{
        socket_type     = stream
        protocol        = tcp
        wait            = no
        user            = root
        server          = /usr/sbin/in.telnetd
}
```

Restart the xinet daemon to have it pick up the new service. Test the telnet service.

```
#  systemctl restart xinetd
#  telnet localhost
```

3. Enable **sshd** in `/etc/hosts.allow`:

```
sshd : all
```

4. Disable all other access in `/etc/hosts.deny`:

```
all : all
```

5. Test the connection to **telnet** (it should fail):

```
$ telnet localhost
```

Exercise 15.2: Enable a firewall which blocks all unwanted traffic

- Ensure SSH traffic is allowed.
- Ensure returning outbound traffic is allowed.
- Ensure all traffic on the loopback interface is allowed.
- Ensure all other traffic is blocked with `DROP`.
- Ensure the firewall rules persist through a reboot.

Solution 15.2

1. Allow all loopback traffic:

```
$ iptables -A INPUT -i lo -j ACCEPT
```

2. Allow all returning traffic:

```
# iptables -A INPUT -m state --state=ESTABLISHED,RELATED -j ACCEPT
```

3. Allow inbound SSH traffic:

```
# iptables -A INPUT -p tcp --dport 22 -j ACCEPT
```

4. Block all other traffic:

```
# iptables -P INPUT DROP
```

5. Save your firewall rules:

   - On **CentOS**:

```
# service iptables save
```

- On **OpenSUSE**, the easiest way to save persistent firewall rules is to use the **YaST** tool.
- On **Ubuntu**:
  (a) Install the package `iptables-persistent`.
  (b) Run this command to store the current rules:
  ```
  # iptables-save >/etc/iptables/rules.v4
  ```


Chapter 16

Virtualization Overview

16.1 Labs

Exercise 16.1: Install and test apache as a Docker application

Install, run and test the **docker** package and the **httpd** container.

Docker requires a **Linux** kernel of 3.10 or greater. If your kernel version is older than 3.10 please skip this exercise.

This exercise requires a 64 bit architecture system.

It is common practice to run containers as a non-root user, but in this exercise we will execute the **docker** commands as root.

Solution 16.1

1. Make sure **Docker** is installed.
   - On **CentOS**:
     ```
     # yum install docker
     ```
   - On **OpenSUSE**:
     ```
     # zypper install docker
     ```
   - On **Ubuntu**:
     ```
     # apt-get install docker.io
     ```
2. Start the **docker** service.
   ```
   # systemctl start docker
   ```
3. Search for the **httpd** container.
   ```
   # docker search apache
   ```
4. Retrieve the container.
   ```
   # docker pull docker.io/httpd
   ```

This may take a couple of minutes while all the components download.

5. List the installed containers.

```
# docker images
```

6. List the components associated with the images.

```
# docker images --all
```

7. Start the **httpd docker** container. The terminal will appear to hang as it is now connected to the **httpd** daemon.

```
# docker run httpd
```

8. Open a web browser with the IP address shown in the **docker httpd** output.

```
# w3m -dump http://172.0.0.1
```

9. Stop the container and the **docker** service from the **docker** window and clean up.

```
# ctl-c
# docker stop httpd
# docker rmi -f docker.io/httpd
# systemctl stop docker
```

Chapter 17

High Availability

17.1 Labs

There is no lab to complete for this chapter.

Chapter 18

System log

18.1 Labs

There is no lab to complete for this chapter.

Chapter 19

Package Management

19.1 Labs

Exercise 19.1: Building and installing from source, using git

stress-ng is a very useful utility for both system administrators and developers. It exercises (stresses) most computer and operating system software and hardware facilities. It has an almost infinite number of options, including over 105 stress tests.

The home page for **stress-ng** is http://kernel.ubuntu.com/~cking/stress-ng.

In this exercise we are going to obtain the source using **git**, the distributed source control system originally developed for use with the **Linux** kernel, but now used by literally millions of projects. We will then compile and install.

1. Obtain the source by **cloning** the **git** repository:

   ```
   $ git clone -v git://kernel.ubuntu.com/cking/stress-ng.git
   ```

   If you do not have **git** installed, do so with your packaging system; modern distributions generally come with it by default.

2. Compile it with:

   ```
   $ cd stress-ng
   $ make
   ```

   You may find the compile fails due to some missing headers, due to a missing development package. For example on a **RHEL 7** system one might get:

   ```
   .........
   cc -O2 -Wall -Wall -Wextra -DVERSION="0.05.00" -O2 -c -o stress-key.o stress-key.c
   stress-key.c:36:22: fatal error: keyutils.h: No such file or directory
    #include <keyutils.h>
                        ^
   compilation terminated.
   make: *** [stress-key.o] Error 1
   ```

   This (and another missing header problem) should be fixed with:

   ```
   $ sudo yum install keyutils-libs-devel libattr-devel
   ```

   On a **Debian**-based system, such as **Ubuntu** you might need:

```
$ sudo apt-get install libkeyutils-dev libattr1-dev
```

On other distributions or versions package names may differ, so happy hunting! This kind of snag is one advantage to using pre-packaged software from distributions.

3. Test with

```
$ ./stress-ng  -c 3 -t 10s -m 4
```

which ties up the system with 3 CPU hogs for 10 seconds while using 1 GB of memory.

4. Install with:

```
$ sudo make install
```

5. Change directories and test again and also see if the documentation was also installed properly.

```
$ cd /tmp
$ stress-ng  -c 3 -t 10s -m 4
$ man stress-ng
```

Note that uninstalling can be a pain, which is one reason we have packaging systems.

Exercise 19.2: Building a Debian package from source

In this exercise we will build a **Debian** package from its upstream source tarball. (Of course if you are on a non-**Debian** based system you can not perform this exercise!)

We will use a simple **hello** program package, the source of which is contained in the `SOLUTIONS` tarball you can obtain from https://training.linuxfoundation.org/cm/LFS211.

Before beginning you may want to make sure you have the necessary utilities installed with:

```
$ sudo apt-get install dh-make fakeroot build-essential
```

The contents are:

```
$ tar xvf myappdebian-1.0.tar.gz
myappdebian-1.0/
myappdebian-1.0/Makefile
myappdebian-1.0/myhello.c
myappdebian-1.0/README
```

where:

```
$ cat README

Some very informative information should go in here :)

$ cat myhello.c

#include <stdio.h>
#include <stdlib.h>
char hello_string[] = "hello world";
int main ()
{
        printf ("\n%s\n\n", hello_string);
        exit (EXIT_SUCCESS);
}

$ cat Makefile
```



```
BIN = $(DESTDIR)/usr/bin
CFLAGS :=        -O $(CFLAGS)
TARGET=                myhello
SRCFILES=        myhello.c

all: $(TARGET)

$(TARGET):        $(SRCFILES)
        $(CC) $(CFLAGS) -o $(TARGET) $(SRCFILES)

install: $(TARGET)
        install -d $(BIN)
        install $(TARGET) $(BIN)

clean:
        rm -f $(TARGET)
```

Note: You can certainly construct a simpler `Makefile`. However, you must have the line:

```
BIN = $(DESTDIR)/usr/bin
```

and point the installation to the `BIN` directory, or the executable will not be installed as part of the package.

The main steps in the following are encapsulated in the `lab_makedeb.sh` which is also in the `SOLUTIONS` tarball for this class section:

1. Make a working directory and put a copy of the source tarball in there:

   ```
   $ rm -rf  WORK && mkdir WORK && cd WORK
   $ cp ../myappdebian-1.0.tar.gz .
   ```

2. Expand the source tarball:

   ```
   $ tar xvf myappdebian-1.0.tar.gz
   ```

3. Go into the expanded directory and build the package, using **dh_make**, one of several possible package builder programs:

   ```
   $ cd myappdebian-1.0
   $ dh_make -f ../*myappdebian-1.0.tar.gz
   $ dpkg-buildpackage -uc -us
   ```

4. Make sure the program works!

   ```
   $ ./myhello
   hello world
   ```

   Verify its contents:

   ```
   $ dpkg --contents ../*.deb
   ```

5. Take a good look at all the files in the `debian` directory and try to imagine building them all by hand!

6. Install the package:

   ```
   $ cd ..
   $ sudo dpkg --install *.deb
   ```

   Verify the installation worked:

   ```
   $ myhello
   hello world
   ```


You can uninstall the package with:

```
$ sudo dpkg --remove myappdebian
```

Exercise 19.3: Building an RPM

In the LFS211_*SOLUTIONS.tar.bz2 file is an example of rpm creation. The SOLUTIONS tar ball contains:

- a source code file, `myapprpm-1.0.0.tar.gz`
- a **SPEC** file, `myapprpm.spec`
- and a script, `myfirstrpm.sh` to aid in the construction of the **rpm** file.

The script can be used to build the **rpm** file or as a guide to create the **rpm** manually. After installing the newly created **rpm** file, test the application by running the "`myhello`" command.