

LFS211

Linux Networking and Administration

Version 2020-11-24



Version 2020-11-24

© Copyright the Linux Foundation 2020. All rights reserved.

© Copyright the Linux Foundation 2020. All rights reserved.

The training materials provided or developed by The Linux Foundation in connection with the training services are protected by copyright and other intellectual property rights.

Open source code incorporated herein may have other copyright holders and is used pursuant to the applicable open source license.

The training materials are provided for individual use by participants in the form in which they are provided. They may not be copied, modified, distributed to non-participants or used to provide training to others without the prior written consent of The Linux Foundation.

No part of this publication may be reproduced, photocopied, stored on a retrieval system, or transmitted without express prior written consent.

Published by:

the **Linux Foundation**

<https://www.linuxfoundation.org>

No representations or warranties are made with respect to the contents or use of this material, and any express or implied warranties of merchantability or fitness for any particular purpose or specifically disclaimed.

Although third-party application software packages may be referenced herein, this is for demonstration purposes only and shall not constitute an endorsement of any of these software applications.

Linux is a registered trademark of Linus Torvalds. Other trademarks within this course material are the property of their respective owners.

If there are any questions about proper and fair use of the material herein, please contact info@linuxfoundation.org.

Contents

| | | |
|-----------|---|-----------|
| 1 | Introduction | 1 |
| 1.1 | Labs | 1 |
| 2 | Linux Networking Concepts and Review | 3 |
| 2.1 | Labs | 3 |
| 3 | Network Configuration | 15 |
| 3.1 | Labs | 15 |
| 4 | Network Troubleshooting and Monitoring | 19 |
| 4.1 | Labs | 19 |
| 5 | Remote Access | 23 |
| 5.1 | Labs | 23 |
| 6 | Domain Name Service | 29 |
| 6.1 | Labs | 29 |
| 7 | HTTP Servers | 41 |
| 7.1 | Labs | 41 |
| 8 | Advanced HTTP Servers | 53 |
| 8.1 | Labs | 53 |
| 9 | Email Servers | 61 |
| 9.1 | Labs | 61 |
| 10 | File Sharing | 69 |
| 10.1 | Labs | 69 |
| 11 | Advanced Networking | 73 |
| 11.1 | Labs | 73 |
| 12 | HTTP Caching | 79 |
| 12.1 | Labs | 79 |
| 13 | Network File Systems | 81 |
| 13.1 | Labs | 81 |
| 14 | Introduction to Network Security | 87 |
| 14.1 | Labs | 87 |
| 15 | Firewalls | 89 |
| 15.1 | Labs | 89 |
| 16 | LXC Virtualization Overview | 95 |
| 16.1 | Labs | 95 |

| | |
|------------------------------|------------|
| 17 High Availability | 101 |
| 17.1 Labs | 101 |
| 18 Database | 103 |
| 18.1 Labs | 103 |
| 19 System log | 107 |
| 19.1 Labs | 107 |
| 20 Package Management | 111 |
| 20.1 Labs | 111 |

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:

```
$ sudo 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



On openSUSE

```
# zypper install vsftpd tnftp
```



On Ubuntu

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



On 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



Please Note

The location of the **vsftpd** command varies depending on the distribution, it may be in `/usr/sbin/vsftpd` or `/sbin/vsftpd`. Check both places.



On CentOS

```
# /sbin/vsftpd /etc/vsftpd/vsftpd.conf &
```



On openSUSE

```
# /sbin/vsftpd /etc/vsftpd.conf &
```



On Ubuntu

```
# /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



Please Note

This section is marked **optional** as some distros do not include the **SYSV** init subsystem by default and the scripts to control the services may not be included in some service packages. It is important to know about the **SYSV** init subsystem as some older applications may not include **systemd** service files.

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 and examine the files installed. Notice the absolute path name of the binary **stress** and the absence of a **stress.service** file in the **systemd** file structure.



Very Important

Some distributions have the **stress** package; others have the newer **stress-ng**. Either package will work as **stress-ng** supports the same command line options as **stress**.



On CentOS

```
# yum install stress
# rpm -ql stress
```



On openSUSE

```
# zypper install stress-ng
# rpm -ql stress-ng
```



On Ubuntu

```
# apt-get install stress
# dpkg -L stress
```

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`. You will require **root** level access to create a file in this directory.

```
$ sudo vim /usr/lib/systemd/system/foo.service
```



/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 256M
[Install]
WantedBy=multi-user.target
```

A copy of this file (`foo1.service`) can be found in the tarball in the `LFS211/SOLUTIONS/s_02/` directory.

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

The example program **stress** which is now a service, does not display much feedback as to what it is doing. The **systemctl status** of the service can be checked, the output would look something like this:

```

File Edit View Search Terminal Help
lee@yoga:~$ sudo systemctl status foo
● foo.service - Example service unit to run stress
   Loaded: loaded (/usr/lib/systemd/system/foo.service; disabled; vendor preset: enabled)
   Active: active (running) since Thu 2018-11-15 10:12:36 EST; 29s ago
 Main PID: 12687 (stress)
    Tasks: 11 (limit: 4915)
   Memory: 213.8M
    CGroup: /system.slice/foo.service
            └─12687 /usr/bin/stress --cpu 4 --io 4 --vm 2 --vm-bytes 256M
              └─12688 /usr/bin/stress --cpu 4 --io 4 --vm 2 --vm-bytes 256M
                └─12689 /usr/bin/stress --cpu 4 --io 4 --vm 2 --vm-bytes 256M
                  └─12690 /usr/bin/stress --cpu 4 --io 4 --vm 2 --vm-bytes 256M
                    └─12691 /usr/bin/stress --cpu 4 --io 4 --vm 2 --vm-bytes 256M
                      └─12692 /usr/bin/stress --cpu 4 --io 4 --vm 2 --vm-bytes 256M
                        └─12693 /usr/bin/stress --cpu 4 --io 4 --vm 2 --vm-bytes 256M
                          └─12694 /usr/bin/stress --cpu 4 --io 4 --vm 2 --vm-bytes 256M
                            └─12695 /usr/bin/stress --cpu 4 --io 4 --vm 2 --vm-bytes 256M
                              └─12696 /usr/bin/stress --cpu 4 --io 4 --vm 2 --vm-bytes 256M
                                └─12697 /usr/bin/stress --cpu 4 --io 4 --vm 2 --vm-bytes 256M

Nov 15 10:12:36 yoga systemd[1]: Started Example service unit to run stress.
Nov 15 10:12:36 yoga stress[12687]: stress: info: [12687] dispatching hogs: 4 cpu, 4 io, 2 vm, 0 hdd
lee@yoga:~$

```

Figure 2.1: **systemctl status foo**

Examining the output of the **top** command will show two processes of **stress** with more than the specified 256M of memory, four processes of **stress** with nearly 100% CPU usage and four processes with neither high CPU or high memory usage. They would be the memory hogs, CPU hogs, and io hogs, respectively. See the example below:

```

File Edit View Search Terminal Help
top - 10:14:39 up 2:39, 1 user, load average: 9.14, 4.06, 2.87
Tasks: 303 total, 10 running, 293 sleeping, 0 stopped, 0 zombie
%Cpu(s): 56.0 us, 43.3 sy, 0.0 ni, 0.0 id, 0.6 wa, 0.0 hi, 0.0 si, 0.0 st
MiB Mem : 15786.5 total, 8109.7 free, 3808.1 used, 3868.7 buff/cache
MiB Swap: 2048.0 total, 2048.0 free, 0.0 used. 10771.2 avail Mem

  PID USER      PR  NI  VIRT  RES  SHR S %CPU  %MEM    TIME+  COMMAND
12691 root        20   0   4056   100    0 R  99.0   0.0   1:59.64 stress
12694 root        20   0   4056   100    0 R  98.7   0.0   2:01.10 stress
12688 root        20   0   4056   100    0 R  98.0   0.0   1:59.32 stress
12696 root        20   0   4056   100    0 R  97.4   0.0   1:59.99 stress
12693 root        20   0 266204 133160 272 R  96.0   0.8   1:59.42 stress
12690 root        20   0 266204 90656 272 R  94.4   0.6   1:59.67 stress
12695 root        20   0   4056   100    0 D  44.0   0.0   0:44.99 stress
12689 root        20   0   4056   100    0 R  43.4   0.0   0:43.68 stress
12692 root        20   0   4056   100    0 R  42.1   0.0   0:43.82 stress
12697 root        20   0   4056   100    0 R  40.7   0.0   0:43.50 stress
 2564 lee        20   0 3767724 424628 158296 S  16.9   2.6   6:57.38 gnome-she+
 447 root        0 -20    0    0    0 I   4.6   0.0   0:12.16 kworker/1+
2396 lee        20   0 866864 208236 154892 S   4.3   1.3   4:39.05 Xorg
 233 root       -51   0    0    0    0 S   2.3   0.0   0:46.09 irq/51-SY+
 300 root        0 -20    0    0    0 I   2.3   0.0   0:04.81 kworker/4+
13093 lee        20   0 508924 36068 28092 S   1.7   0.2   0:00.55 gnome-scr+
  68 root        25   5    0    0    0 S   1.0   0.0   0:37.74 ksm

```

Figure 2.2: **top**

As we are interested specifically in the **stress** service and its child processes, we provide a script for monitoring the service processes running.

track-STRESS.sh looks for a **stress** process with a PPID of 1, then all of the related child processes. Once the processes are located some data is extracted with the **ps** command.

A copy of this script ([track-STRESS.sh](#)) can be found in the tarball in the [LFS211/SOLUTIONS/s_02/](#) directory.

SH

track-STRESS.sh

```
#!/bin/bash
#
# This little script is used with the LFS311 lab exercise
# on systemd startup files and their affect on a background
# service. The "foo" service.
#
# The script looks for "stress" launched with a PPID of 1
# and it's children processes then uses the "ps" command
# to collect some information.

while [ true ]
do

# reset the variables to make sure we are picking up
# current information
PID=""
PID1=""
clear

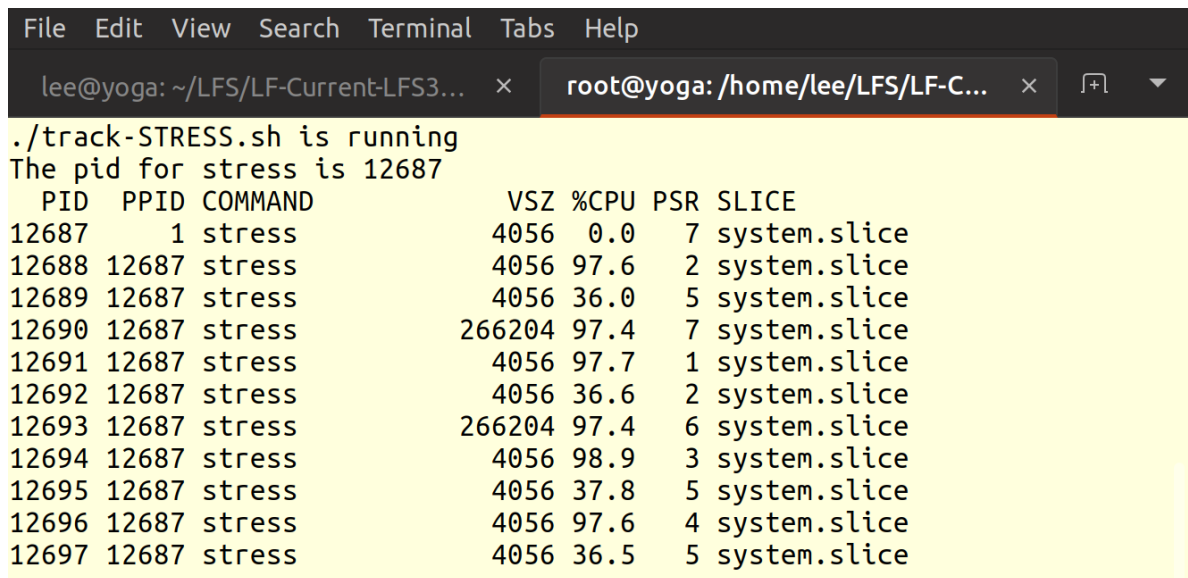
echo "$0 is running"

# sift and sort the pid's
PID1=`ps -ef | grep stress | grep -v grep | awk '{print $1,$2,$3,$6}'`
PID=`echo $PID1 | grep "1 "| awk '{print $2}'`
echo "The pid for stress is $PID"

# using our list of pid's, grab some information
ps --ppid $PID --pid $PID -o pid,ppid,comm,vsz,pcpu,psr,slice 2>/dev/null

sleep 5
done
exit
```

An example of the script running:



```
File Edit View Search Terminal Tabs Help
lee@yoga: ~/LFS/LF-Current-LFS3... x root@yoga: /home/lee/LFS/LF-C... x [+]
```

```
./track-STRESS.sh is running
The pid for stress is 12687
```

| PID | PPID | COMMAND | VSZ | %CPU | PSR | SLICE |
|-------|-------|---------|--------|------|-----|--------------|
| 12687 | 1 | stress | 4056 | 0.0 | 7 | system.slice |
| 12688 | 12687 | stress | 4056 | 97.6 | 2 | system.slice |
| 12689 | 12687 | stress | 4056 | 36.0 | 5 | system.slice |
| 12690 | 12687 | stress | 266204 | 97.4 | 7 | system.slice |
| 12691 | 12687 | stress | 4056 | 97.7 | 1 | system.slice |
| 12692 | 12687 | stress | 4056 | 36.6 | 2 | system.slice |
| 12693 | 12687 | stress | 266204 | 97.4 | 6 | system.slice |
| 12694 | 12687 | stress | 4056 | 98.9 | 3 | system.slice |
| 12695 | 12687 | stress | 4056 | 37.8 | 5 | system.slice |
| 12696 | 12687 | stress | 4056 | 97.6 | 4 | system.slice |
| 12697 | 12687 | stress | 4056 | 36.5 | 5 | system.slice |

Figure 2.3: track-STRESS.sh

Since `/usr/lib/systemd/system/foo.service` is the default configuration supplied by the packager of the service and may be altered by the vendor at any time, create a custom unit file in `/etc/systemd/system/foo.service` for the **stress** service. This file is not usually overwritten by the vendor so local customizations can go here. Change the parameters slightly for the **foo** service using this directory. It is common practice to copy the vendor unit file into the `/etc/systemd/system/` directory and make appropriate customizations.



`/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 256M
[Install]
WantedBy=multi-user.target
```

A copy of this file (`foo2.service`) can be found in the tarball in the `LFS211/SOLUTIONS/s_02/` directory.

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

```
# track-STRESS.sh
# systemctl status foo -l
# systemd-delta
```

The changes to the configuration file can be seen with the **track-STRESS.sh** script, notice the number of memory hogs is now 4 and the CPU hogs is reduced to 2.

```

File Edit View Search Terminal Help
./track-STRESS.sh is running
The pid for stress is 17860
  PID PPID COMMAND      VSZ %CPU PSR SLICE
17860   1 stress           4056 0.0  4 system.slice
17861 17860 stress           4056 99.4  0 system.slice
17862 17860 stress           4056 43.7  1 system.slice
17863 17860 stress        266204 99.3  2 system.slice
17864 17860 stress           4056 99.5  5 system.slice
17865 17860 stress           4056 43.9  1 system.slice
17866 17860 stress        266204 99.4  4 system.slice
17867 17860 stress        266204 99.3  3 system.slice
17868 17860 stress        266204 99.3  6 system.slice

```

Figure 2.4: track-STRESS.sh with new unit file

Which configuration (or **unit** file) file is active is not clear in the script. Use `systemctl status foo` to see which unit file is being used. This will show which configuration files are being used but not the differences in the files.

```

File Edit View Search Terminal Help
lee@yoga:~$ sudo systemctl status foo
● foo.service - Example service unit to run stress
   Loaded: loaded (/etc/systemd/system/foo.service; disabled; vendor preset: enabled)
   Active: active (running) since Thu 2018-11-15 11:36:45 EST; 1min 17s ago
 Main PID: 17860 (stress)
    Tasks: 9 (limit: 4915)
   Memory: 482.7M
    CGroup: /system.slice/foo.service
            └─17860 /usr/bin/stress --cpu 2 --io 2 --vm 4 --vm-bytes 256M
              └─17861 /usr/bin/stress --cpu 2 --io 2 --vm 4 --vm-bytes 256M
                └─17862 /usr/bin/stress --cpu 2 --io 2 --vm 4 --vm-bytes 256M
                  └─17863 /usr/bin/stress --cpu 2 --io 2 --vm 4 --vm-bytes 256M
                    └─17864 /usr/bin/stress --cpu 2 --io 2 --vm 4 --vm-bytes 256M
                      └─17865 /usr/bin/stress --cpu 2 --io 2 --vm 4 --vm-bytes 256M
                        └─17866 /usr/bin/stress --cpu 2 --io 2 --vm 4 --vm-bytes 256M
                          └─17867 /usr/bin/stress --cpu 2 --io 2 --vm 4 --vm-bytes 256M
                            └─17868 /usr/bin/stress --cpu 2 --io 2 --vm 4 --vm-bytes 256M

Nov 15 11:36:45 yoga systemd[1]: Started Example service unit to run stress.
Nov 15 11:36:45 yoga stress[17860]: stress: info: [17860] dispatching hogs: 2 cpu, 2 io, 4 vm, 0 hdd
lee@yoga:~$

```

Figure 2.5: systemctl status foo changing the unit file

To see the details of the unit file changes the **systemd-delta** command can be used. The output has the changes in **diff** format, making it easy to see what has changed. See the example below:

```

File Edit View Search Terminal Help

[OVERRIDDEN] /etc/systemd/system/foo.service → /usr/lib/systemd/system/foo.service

--- /usr/lib/systemd/system/foo.service 2018-11-15 09:24:28.183557249 -0500
+++ /etc/systemd/system/foo.service      2018-11-15 11:34:21.689073568 -0500
@@ -1,6 +1,6 @@
[Unit]
Description=Example service unit to run stress
[Service]
-ExecStart=/usr/bin/stress --cpu 4 --io 4 --vm 2 --vm-bytes 256M
+ExecStart=/usr/bin/stress --cpu 2 --io 2 --vm 4 --vm-bytes 256M
[Install]
WantedBy=multi-user.target

[EXTENDED] /lib/systemd/system/rc-local.service → /lib/systemd/system/rc-local.service.d/debian.con
[EXTENDED] /lib/systemd/system/user@.service → /lib/systemd/system/user@.service.d/timeout.conf

4 overridden configuration files found.
lines 8-25/25 (END)

```

Figure 2.6: **systemd-delta** showing unit file override

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 our **stress** service and verify the changes are active. Our example file for `foo.service` using a drop-in directory (`00-foo.conf`) can be found in the SOLUTIONS tarball in the [LFS211/SOLUTIONS/s_02/](#) directory and contains:



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

```

[Service]
ExecStart=
ExecStart=/usr/bin/stress --cpu 1 --vm 1 --io 1 --vm-bytes 128M

```

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

```

# track-STRESS.sh
# systemctl status foo -l
# systemd-delta

```

The information in the drop in file over writes the unit file. In this example the number of “hogs” has been greatly reduced.

```

File Edit View Search Terminal Help

./track-STRESS.sh is running
The pid for stress is 8044
  PID  PPID  COMMAND      VSZ  %CPU  PSR  SLICE
  8044    1  stress       4056  0.0   3  system.slice
  8045  8044  stress       4056  99.9   1  system.slice
  8046  8044  stress       4056  31.4   6  system.slice
  8047  8044  stress     135132  99.8   7  system.slice

```

Figure 2.7: **track-STRESS.sh** using dropin file

systemctl status shows the dropin file is active. If there was several dropin files it would show the order that were applied to

the service.

```
File Edit View Search Terminal Help
root@yoga:/etc/systemd/system/foo.service.d# systemctl status foo
● foo.service - Example service unit to run stress
   Loaded: loaded (/etc/systemd/system/foo.service; disabled; vendor preset: enabled)
   Drop-In: /etc/systemd/system/foo.service.d
            └─00-foo.conf
   Active: active (running) since Thu 2018-11-15 14:06:05 EST; 13s ago
   Main PID: 8044 (stress)
     Tasks: 4 (limit: 4915)
    Memory: 103.5M
     CGroup: /system.slice/foo.service
             └─8044 /usr/bin/stress --cpu 1 --vm 1 --io 1 --vm-bytes 128M
               └─8045 /usr/bin/stress --cpu 1 --vm 1 --io 1 --vm-bytes 128M
                 └─8046 /usr/bin/stress --cpu 1 --vm 1 --io 1 --vm-bytes 128M
                   └─8047 /usr/bin/stress --cpu 1 --vm 1 --io 1 --vm-bytes 128M

Nov 15 14:06:05 yoga systemd[1]: Started Example service unit to run stress.
Nov 15 14:06:05 yoga stress[8044]: stress: info: [8044] dispatching hogs: 1 cpu, 1 io, 1 vm, 0 hdd
root@yoga:/etc/systemd/system/foo.service.d#
```

Figure 2.8: **systemctl status** showing unit file override and dropin file

Like the other commands, **systemd-delta** shows the files used by the service. In addition to the files used, the details of the changes in the unit file are displayed. Notice the changes to the service made with the drop-in file are not displayed, only the file name.

```
File Edit View Search Terminal Help

[OVERRIDDEN] /etc/systemd/system/foo.service → /usr/lib/systemd/system/foo.service

--- /usr/lib/systemd/system/foo.service 2018-11-15 09:24:28.183557249 -0500
+++ /etc/systemd/system/foo.service      2018-11-15 11:34:21.689073568 -0500
@@ -1,6 +1,6 @@
[Unit]
Description=Example service unit to run stress
[Service]
-ExecStart=/usr/bin/stress --cpu 4 --io 4 --vm 2 --vm-bytes 256M
+ExecStart=/usr/bin/stress --cpu 2 --io 2 --vm 4 --vm-bytes 256M
[Install]
WantedBy=multi-user.target

[EXTENDED] /lib/systemd/system/rc-local.service → /lib/systemd/system/rc-local.service.d/debian.conf
[EXTENDED] /lib/systemd/system/user@.service → /lib/systemd/system/user@.service.d/timeout.conf

4 overridden configuration files found.
lines 8-25/25 (END)
```

Figure 2.9: **systemd-delta** showing unit file and dropin file

With **systemd**, additional features and capabilities can be easily added. As an example, **cgroups** controls can be added to our 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:



/etc/systemd/system/foo.slice

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


A copy of this file (`foo.slice`) can be found in the tarball in the `LFS211/SOLUTIONS/s_02/` directory.

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`



connect slice file to service file

```
Slice=foo.slice
```

Restart the services and examine the differences with:

```
# systemctl stop foo
# systemctl daemon-reload
# systemctl start foo
# systemctl status foo -l
# systemd-delta
# top
# track-STRESS.sh
```

The cgroup information in the **slice** has been applied to the service. Notice the amount of CPU resource consumed. The total is 30% of one processor but it may be spread across multiple CPU's.

| File Edit View Search Terminal Help | | | | | | |
|-------------------------------------|------|---------|--------|------|-----|-----------|
| ./track-STRESS.sh is running | | | | | | |
| The pid for stress is 8989 | | | | | | |
| PID | PPID | COMMAND | VSZ | %CPU | PSR | SLICE |
| 8989 | 1 | stress | 4056 | 0.0 | 7 | foo.slice |
| 8990 | 8989 | stress | 4056 | 12.7 | 5 | foo.slice |
| 8991 | 8989 | stress | 4056 | 5.0 | 4 | foo.slice |
| 8992 | 8989 | stress | 135132 | 12.0 | 3 | foo.slice |

Figure 2.10: track-STRESS.sh using slice attribute

The **systemctl status** command shows the change in **CGroup** with the **slice** attribute active.

```
File Edit View Search Terminal Help
root@yoga:/etc/systemd/system# systemctl status foo
● foo.service - Example service unit to run stress
   Loaded: loaded (/etc/systemd/system/foo.service; disabled; vendor preset: enabled)
   Drop-In: /etc/systemd/system/foo.service.d
            └─00-foo.conf
   Active: active (running) since Thu 2018-11-15 14:30:29 EST; 2min 53s ago
 Main PID: 8989 (stress)
    Tasks: 4 (limit: 4915)
   Memory: 10.4M
   CGroup: /foo.slice/foo.service
            └─8989 /usr/bin/stress --cpu 1 --vm 1 --io 1 --vm-bytes 128M
              └─8990 /usr/bin/stress --cpu 1 --vm 1 --io 1 --vm-bytes 128M
                └─8991 /usr/bin/stress --cpu 1 --vm 1 --io 1 --vm-bytes 128M
                  └─8992 /usr/bin/stress --cpu 1 --vm 1 --io 1 --vm-bytes 128M

Nov 15 14:30:29 yoga systemd[1]: Started Example service unit to run stress.
Nov 15 14:30:29 yoga stress[8989]: stress: info: [8989] dispatching hogs: 1 cpu, 1 io, 1 vm, 0 hdd
root@yoga:/etc/systemd/system#
```

Figure 2.11: systemctl status showing CGroup and slice

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)**, **systemd-delta(1)** 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
```

In some of the later editions of the distribution's **systemd** may have an optional feature **systemd-resolved.service** active. Check to see if **systemd-resolved** is active and if so record the configuration in `/etc/systemd/resolved`.

Check to see if **systemd-resolved** is active:

```
# systemctl status systemd-resolved
```

If **systemd-resolved** is active, examine the override file:

```
# cat /etc/systemd/resolved.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.

**Very Important**

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 `/etc/network/interfaces` and add or modify the configuration like below, using your discovered values:

**On Debian, Ubuntu, or Linux Mint****`/etc/network/interfaces`**

```
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
```

A copy of this file (`interfaces`) can be found in the tarball in the `LFS211/SOLUTIONS/s_03/` directory.

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

**On RedHat, Centos, or Fedora****`/etc/sysconfig/network-scripts/ifcfg-<adapter name>`**

```
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 openSUSE**

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



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

```
NETCONFIG_DNS_STATIC_SERVERS=8.8.8.8
```

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

Chapter 4

Network Troubleshooting and Monitoring



4.1 Labs

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

1. 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. Confirm either **iproute** or **iproute2** is installed on your system.
2. Use the `netem` option of the **tc** command to introduce a network problem on the server (random packet drops).

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



Please Note

If your version of **tc** does not support the `loss random 40` option substitute `corrupt 30%`. The new command would be:

```
# 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.



Please Note

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



On Debian, Ubuntu, or Linux Mint

Debian 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 **netstat** to prove where the SMTP daemon is listening.

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

or

```
# ss -lnt
```

- Use **telnet** 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: OPTIONAL: Block traffic to a service with TCP Wrappers and prove it is blocked



Very Important

The support for **tcp-wrappers** in **vsftpd** has been removed in many distributions causing this lab to fail.

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

✓ Solution 4.3



On Debian, Ubuntu, or Linux Mint

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

```
tcp_wrappers=yes
```

1. Start a service:

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


2. Check the port with **telnet**:

```
$ telnet localhost ftp
```

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

```
vsftpd: ALL
```



On openSUSE

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 with:

```
# 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.



Please 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 from release to release.

Check if the parameter `PermitRootLogin` is set to `yes` in `/etc/ssh/sshd_config`; 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 `$HOME/.ssh/config` and add the following contents:



`$HOME/.ssh/config`

```
host garply
hostname localhost
user root

host *
ForwardX11 yes
```

2. Verify or update the permissions on `$HOME/.ssh/config` to allow read and write for the file owner only.

```
$ chmod 600 $HOME/.ssh/config
```

3. 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 `/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 `/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
```



Please 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.

pssh (or **parallel-ssh**) 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 `~/.ssh/knownhosts`. The **pssh** commands are most secure with the ssh key copied into the target's `authorized_keys` file.



Please 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:



On Debian, Ubuntu, or Linux Mint

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

Verify the program names

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



On RedHat, Centos, or Fedora

```
$ sudo yum install pssh

Verify the program names

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



On 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 password-less connection, if you are prompted for a password 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 and test a VNC server

Install VNC server.

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

Solution 5.6

1. Ensure VNC server and viewer are installed.



On RedHat, Centos, or Fedora

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



On Debian, Ubuntu, or Linux Mint

- **Ubuntu 16.04**

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

- **Ubuntu 18.04 and later**, use tigervnc.

```
# apt-get purge tightvncserver
```

```
# apt-get install tigervnc-standalone-server tigervnc-common tigervnc-viewer
```



On openSUSE

```
# zypper install tigervnc tigervnc-x11vnc
```

2. Start the server.

```
$ vncserver
```

**Please Note**

The **tigervncserver** defaults only listen to the localhost, unless an additional option is passed to the vncserver when launched. This behavior may be overridden via a configuration file if required or desired. See the man pages for **vncserver** and **vnc.conf** for details.

```
$ vncserver --localhost no
```

The vncserver can also use the “forcing ssh via” connection which requires a ssh connection to the server, and then accesses the vncserver **via** the localhost.

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

```
$ vncviewer localhost:1
```

**Hints:**

- Ensure the `$HOME/.vnc/xstartup` file is executable.
- `$HOME/.vnc/xstartup` may contain references to applications that are not installed; install them.
- This is copy of a **xstartup** file that can be used.

**Sample xstartup script: `$HOME/.vnc/xstartup`**

```
#!/bin/sh

unset SESSION_MANAGER
unset DBUS_SESSION_BUS_ADDRESS
xsetroot -solid grey
xterm -geometry 80x24+0+0 &
xeyes -geometry 100x100+0+0 &
exec twm &
```

(A copy of this file (`xstartup`) can be found in the tarball in the `LFS211/SOLUTIONS/s_05/` directory.)

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
```

Exercise 5.8: VNC Server Auto-start

The VNC service can be automatically started via **systemd**. Some distros have **systemd** parameter files but since VNC requires a port number and user id customization is required. Create a **systemd** configuration file that will allow two VNC sessions one on port 42 and the other on port 43. The two VNC instances will be for the user student and be encrypted.

Solution 5.8

1. Create the **systemd** configuration file `/etc/systemd/system/vncserver@.service` with the following content:



`/etc/systemd/system/vncserver@.service`

```
[Unit]
Description=Remote desktop service (VNC) on port :%I
After=syslog.target network.target

[Service]
Type=forking
User=student

ExecStartPre=/usr/bin/vncserver -kill :%i
ExecStart=/usr/bin/vncserver :%i -localhost
PIDFile=/home/student/.vnc/%H:%i.pid
ExecStop=/usr/bin/vncserver -kill :%i

[Install]
WantedBy=multi-user.target
```

A copy of this file (`vncserver@.service`) can be found in the tarball in the [LFS211/SOLUTIONS/s_05/](#) directory.

2. Re-load the **systemd** configuration files:

```
# systemctl daemon-reload
```

3. Start the VNCservers:

```
# systemctl start vncserver@42
# systemctl start vncserver@43
```

4. Verify the VNC servers are working:

```
$ vncviewer -via student@hostname localhost:42
$ vncviewer -via student@hostname localhost:43
```

5. Make the VNC servers start at boot time:

```
# systemctl enable vncserver@42
# systemctl enable vncserver@43
```


Chapter 6

Domain Name Service



6.1 Labs

Exercise 6.1: Configure Caching DNS



Please Note



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

1. Install the Domain Name Server
2. Configure the Domain Name server for local caching
3. Start the Domain Name Server
4. Test

Solution 6.1

The locations of the configuration files and the content layouts are different depending on the distributions. All of the configurations follow the **bind** guidelines but there is little or no fixed locations. The most common configuration files are **named.conf** and the **zone** files. The included chart shows the locations for **CentOS-8** and **Ubuntu-2004**.

Table 6.1: Location of named files

| File |  CentOS |  Ubuntu |
|---------------------------|--|--|
| named.conf | /etc/named.conf | /etc/bind/named.conf |
| named.conf.options | | /etc/bind/named.conf.options |
| named.conf.local | | /etc/bind/named.conf.local |
| ZONE files | /var/named/ | /etc/bind/ |



On Debian, Ubuntu, or Linux Mint

Install the named server:

```
# apt-get install bind9
```

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



`/etc/bind/named.conf.options`

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

Start the server:

```
# systemctl start bind9
```



On RedHat, Centos, or Fedora

Install the named server:

```
# yum install bind
```

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



`/etc/named.conf`

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

Start the server:

```
# systemctl start named
```



On openSUSE

Install the named server:

```
# zypper install bind
```

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



`/etc/named.conf`

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

Create an empty include file to satisfy the default configuration options.

```
# touch /etc/named.conf.include
```

Start the server:



```
# systemctl start named
```

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



On RedHat, Centos, Fedora, or openSUSE

1. Edit file `/etc/named.conf`, and add a stanza like this:



Add the new zone to named.conf

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



On Debian, Ubuntu, or Linux Mint

2. Edit `/etc/bind/named.conf.local`. Add a stanza like this:



`/etc/bind/named.conf.local`

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

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



Locations of zone files

- 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/`.



example.com.zone, zone file

```
$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.$
```

A copy of this file (`example.com.zone`) can be found in the tarball in the [LFS211/SOLUTIONS/s_06/](#) directory.

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

```
# systemctl restart named
```



On Debian, Ubuntu, or Linux Mint

The command for **Ubuntu** is:

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

6. 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 `named.conf` for your new zone:



On RedHat, Centos, Fedora, or openSUSE

Edit `/etc/named.conf`. Add a stanza like this:



`/etc/named.conf`

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



On Debian, Ubuntu, or Linux Mint

Edit `/etc/bind/named.conf.local`. Add a stanza like this:



`/etc/bind/named.conf.local`

```
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.



reverse zone file

- 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/`



45.20.10.in-addr.arpa.zone, reverse zone file

```
$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.
```

A copy of this file (`45.20.10.in-addr.arpa.zone`) can be found in the tarball in the `LFS211/SOLUTIONS/s_06/` directory.

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
```

Exercise 6.4: Create a view.

This exercise is going to modify the configuration to employ **DNS View's**. A different IP address will be returned depending on the subnet of the requester.

1. Determine the addresses available.
2. Create a **zone** file for the **external** view.
3. Update the `named.conf` to support the new **zone**.
4. Add **views** and network based restrictions to **ALL** the zones.
5. Verify different addresses are returned by the DNS when accessing it by different networks.

Solution 6.4

1. The configuration is going to be IP Address sensitive. Please record your IP addresses.



```
# ip a | grep "inet "
```

```
1 inet 127.0.0.1/8 scope host lo
2 inet 192.168.122.46/24 brd 192.168.122.255 scope global dynamic noprefixroute enp1s0
```



Please Note

For this solution these addresses will be used, please adjust for your installation.

2. Prepare the **zone** file for use as the **external** access. Copy the existing:
 -  **CentOS**- file `/var/named/example.com.zone` to `/var/named/example.com.zone-x` or
 -  **Ubuntu**- file `/etc/bind/example.com.zone` file to `/etc/bind/example.com.zone-x`.
3. The **external** zone file, `example.com.zone-x` is a trimmed version of the original file and the addresses have been changed.



example.com.zone-x

```
$TTL 30
@ IN SOA localhost. admin.example.com. (
2020040901 ; serial YYYYMMDDRR format
3H ; refresh
1H ; retry
2H ; expire
1M) ; neg ttl

      IN NS localhost.;
www.example.com. IN A 10.0.0.192
foo.example.com. IN A 10.0.0.193
```



```
bar.example.com. IN CNAME www.example.com.
```

A copy of this file (`example.com.zone-x`) can be found in the tarball in the [LFS211/SOLUTIONS/s_06/](#) directory.

4. Update the `named.conf` adding **views**.



Please Note

ALL zone stanzas must have views or no stanzas can have views. Multiple zone stanzas can be in a single view stanza; this example uses a unique view stanza for each zone definition.



Very Important

The following files can be dropped in and will replace the original files that have been used thus far. The names of the files may require adjustment and the original should be saved in case a rewind is required. The files are available in the [LFS211/SOLUTIONS/s_06/](#) directory.



On RedHat, Centos, or Fedora



updated named.conf with view stanzas, named.conf-view

```
options {
    listen-on port 53 { any; };
    listen-on-v6 port 53 { ::1; };
    directory      "/var/named";
    dump-file      "/var/named/data/cache_dump.db";
    statistics-file "/var/named/data/named_stats.txt";
    memstatistics-file "/var/named/data/named_mem_stats.txt";
    secroots-file  "/var/named/data/named.secrets";
    recursing-file  "/var/named/data/named.recursing";
    allow-query    { any; };
    recursion yes;
    dnssec-enable yes;
    dnssec-validation yes;

    managed-keys-directory "/var/named/dynamic";

    pid-file "/run/named/named.pid";
    session-keyfile "/run/named/session.key";

    include "/etc/crypto-policies/back-ends/bind.config";
};

logging {
    channel default_debug {
        file "data/named.run";
        severity dynamic;
    };
};

view "internal" {
    match-clients {127.0.0.0/8; 10.0.3.0/16; };
    zone "example.com." IN {
        type master;
        file "example.com.zone";
    };
};

view "external" {
    match-clients { any; };
    recursion no;
    zone "example.com." IN {
        type master;
        file "example.com.zone-x";
    };
};

view "hint" {
    zone "." IN {
        type hint;
        file "named.ca";
    };
};

view "rfc-zones"{
    include "/etc/named.rfc1912.zones";
};
include "/etc/named.root.key";
```




On Debian, Ubuntu, or Linux Mint



named.conf with views, named.conf-view.ubuntu

```
// This is the primary configuration file for the BIND DNS server named.
//
// Please read /usr/share/doc/bind9/README.Debian.gz for information on the
// structure of BIND configuration files in Debian, *BEFORE* you customize
// this configuration file.
//
// If you are just adding zones, please do that in /etc/bind/named.conf.local

include "/etc/bind/named.conf.options";
include "/etc/bind/named.conf.local";
view "stuff" {
include "/etc/bind/named.conf.default-zones";
};
```



named.conf.options with views, named.conf.options-view.ubuntu

```
options {
    directory "/var/cache/bind";

    // If there is a firewall between you and nameservers you want
    // to talk to, you may need to fix the firewall to allow multiple
    // ports to talk. See http://www.kb.cert.org/vuls/id/800113

    // If your ISP provided one or more IP addresses for stable
    // nameservers, you probably want to use them as forwarders.
    // Uncomment the following block, and insert the addresses replacing
    // the all-0's placeholder.

    // forwarders {
    //     0.0.0.0;
    // };

    //=====
    // If BIND logs error messages about the root key being expired,
    // you will need to update your keys. See https://www.isc.org/bind-keys
    //=====
    dnssec-validation auto;

    //    listen-on-v6 { any; };

    listen-on port 53 { any; };
    allow-query { any; };
    recursion no;

};
```



named.conf.local with views, named.conf.local-view.ubuntu

```
view "internal" {
    match-clients {127.0.0.0/8; 10.0.3.0/24; };
    zone "example.com." IN {
        type master;
        file "/etc/bind/example.com.zone";
    };
};

view "external" {
    match-clients { any; };
    recursion no;
    zone "example.com." IN {
        type master;
        file "/etc/bind/example.com.zone-x";
    };
};

view "rfc-zones"{
    include "/etc/bind/zones.rfc1918";
};
```

5. restart **bind**

```
# systemctl restart named
```

6. Test the new configuration.

The configuration as created should respond with IP Addresses depending on the address that is used. The "dig hostname @nameserver" allows us to specify which address and network to use.

A detailed response is shown here:

```
[lee@localhost ~]$ dig www.example.com @127.0.0.1
```

```
1 ; <<>> DiG 9.11.4-P2-RedHat-9.11.4-26.P2.el8 <<>> www.example.com @127.0.0.1
2 ;; global options: +cmd
3 ;; Got answer:
4 ;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 60660
5 ;; flags: qr aa rd ra; QUERY: 1, ANSWER: 1, AUTHORITY: 1, ADDITIONAL: 1
6
7 ;; OPT PSEUDOSECTION:
8 ; EDNS: version: 0, flags:; udp: 4096
9 ; COOKIE: ebb6acc89fcfc32725ad9a675e8f70e29d79ccae794cfed4 (good)
10 ;; QUESTION SECTION:
11 ;www.example.com.                IN      A
12
13 ;; ANSWER SECTION:
14 www.example.com.                30      IN      A      192.168.111.45
15
16 ;; AUTHORITY SECTION:
17 example.com.                    30      IN      NS      localhost.
18
19 ;; Query time: 1 msec
20 ;; SERVER: 127.0.0.1#53(127.0.0.1)
21 ;; WHEN: Thu Apr 09 15:00:50 EDT 2020
22 ;; MSG SIZE rcvd: 111
```

7. The table below shows the patterns tested and the results. Extra spaces are included in the table for your entries.

Table 6.2: **dig** responses for views

| Command | Answer |
|---|----------------|
| dig www.example.com @127.0.0.1 | 192.168.111.45 |
| dig www.example.com @192.168.122.46 | 10.0.0.192 |
| dig foo.example.com @127.0.0.1 | 192.168.121.11 |
| dig foo.example.com @192.168.122.46 | 10.0.0.193 |
| dig bar.example.com @127.0.0.1 | 192.168.111.45 |
| dig bar.example.com @192.168.122.46 | 10.0.0.192 |
| dig theworld.example.com @127.0.0.1 | 192.74.137.5 |
| dig theworld.example.com @192.168.122.46 | fail |
| dig host42.example.com @127.0.0.1 | 10.20.45.42 |
| dig host42.example.com @192.168.122.46 | fail |
| | |
| | |
| | |
| | |
| | |
| | |
| | |

Chapter 7

HTTP Servers



7.1 Labs

Exercise 7.1: Overview of lab environment

The Apache web server configuration consists of several configuration files, data files, optional directory security and control files. Each of the common distributions use a slightly different layout for the files and directory locations. Apache is the same regardless of the locations used. The examples are generally **CentOS** based with text instructions for the other distros.

The server architecture used is for convince and illustration, not necessarily a production environment.

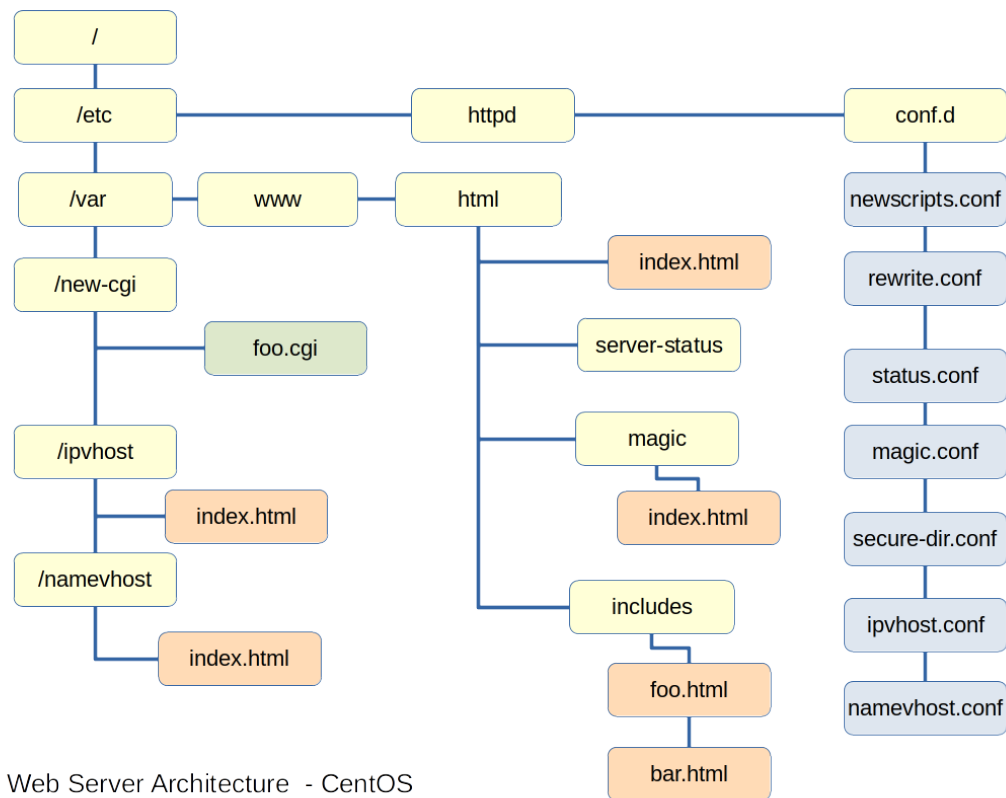


Figure 7.1: top

✍ Exercise 7.2: Install Apache and test.

In this section we will:

- install **apache** and **lynx**
- the directory to create the file in the **DocumentRoot** location which is listed below.
- create a `index.html` file.
- the `index.html` file will contain text indicating it is the **default** web server.
- use a text based web browser to verify the web server is functioning.

✓ Solution 7.2

1. Make sure **Apache** is installed:



On RedHat, Centos, or Fedora

```
# yum install httpd mod_ssl lynx
```



On openSUSE

```
# zypper install apache2 lynx
```



On Debian, Ubuntu, or Linux Mint

```
# apt-get install apache2 lynx
```

2. Create an **index.html** file to serve with **Apache** in the default DocumentRoot.

(a) The location of the **DocumentRoot** directory is different on the distributions. Use the appropriate directory.

The default DocumentRoot directories are:



On RedHat, Centos, or Fedora

```
/var/www/html/
```



On openSUSE

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



On Debian, Ubuntu, or Linux Mint

```
/var/www/html/
```

(b) The contents of **index.html** should be:



index.html

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

(c) A copy of this file (**index.html**) can be found in the tarball in the [LFS211/SOLUTIONS/s_07/](#) directory.

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://127.0.0.1/index.html
```

or

```
$ lynx -dump http://127.0.0.1/index.html
```

Exercise 7.3: Create a new IP virtual host.

**Please Note**

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

- Create an IP alias in the network 192.168.153.0/24.
- 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 7.3

1. Create a temporary IP alias for your main Ethernet. Note the network adapter may be different.

```
# ip addr add 192.168.153.X/24 dev enp1s0
```

Where X is a number no one else in the same LAN is using.

2. Add this new address to `/etc/hosts` with the host name of `ipvhost.example.com` for ease of use later.
3. Create a new directory `/ipvhost/`:

```
# mkdir /ipvhost/
```

4. Create a file named `/ipvhost/index.html`:

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

The file should contain following:

**/ipvhost/index.html**

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

5. If **SELinux** is in **permissive** or **enforcing** mode, verify the **SELinux** permissions are correct:

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

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

**On RedHat, Centos, or Fedora**

```
/etc/httpd/conf.d/ipvhost.conf
```

**On openSUSE**

```
/etc/apache2/vhosts.d/ipvhost.conf
```


**On Debian, Ubuntu, or Linux Mint**`/etc/apache2/sites-enabled/ipvhost.conf`**/etc/httpd/conf.d/ipvhost.conf**

```

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

```

7. Restart apache:

```
# systemctl restart httpd
```

**Please Note**

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

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

```
$ lynx -dump http://127.0.0.1/index.html
```

```
1 This is my default file
```

```
$ lynx -dump http://ipvhost.example.com/index.html
```

```
1 This is my IP vhost
```

✍ Exercise 7.4: 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:

**/namevhost/index.html**

```

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

```

✓ Solution 7.4

1. Create a new name based virtual host definition with the name **namevhost.conf** in the default configuration directory previously used. Create a new config file with the following contents:



namevhost.conf

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

A copy of this file (**namevhost.conf**) can be found in the tarball in the [LFS211/SOLUTIONS/s_07/](#) directory. The solution sample may need changes for your distribution.

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 # or apache2 on Ubuntu or OpenSUSE
```

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

✍ Exercise 7.5: 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 7.5

1. Create the new secure folder:



On RedHat, Centos, or Fedora

```
/var/www/html/secure/
```



On Debian, Ubuntu, or Linux Mint

```
/var/www/html/secure/
```

**On openSUSE**`/srv/www/htdocs/secure/`

2. Create the following stanza in the location listed below to password protect the directory:

**secure-dir.conf**

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

A copy of this file (`secure-dir.conf`) can be found in the tarball in the `LFS211/SOLUTIONS/s_07/` directory.

**On RedHat, Centos, or Fedora**`/etc/httpd/conf.d/secure-dir.conf`**On openSUSE**`/etc/apache2/vhosts.d/secure-dir.conf`**On Debian, Ubuntu, or Linux Mint**`/etc/apache2/sites-enabled/secure-dir.conf`**Please Note**

The Apache modules are not available by default. Either copy the module `/etc/apache2/mods-available/auth_basic.load` to the `/etc/apache2/mods-enabled/` directory or use the command `a2enmod auth_basic`

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

**Please Note**

You may have to install `apache2-utils` if `htpasswd` does not exist.

**On RedHat, Centos, or Fedora**`# htpasswd -c /etc/httpd/secure.users bob`

**On openSUSE**

```
# htpasswd2 /srv/www/secure.users -c bob
```

**On Debian, Ubuntu, or Linux Mint**

On **Ubuntu** use the file:

```
# htpasswd -c /etc/apache2/secure.users bob
```

4. Restart apache:

```
# systemctl restart httpd
```

or

```
# systemctl restart apache2
```

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

✍ Exercise 7.6: 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 7.6

1. Back up the original private key, if one exists:

**On RedHat, Centos, or Fedora**

```
# mv /etc/pki/tls/private/localhost.key /etc/pki/tls/private/localhost.key.orig
```

**On Debian, Ubuntu, or Linux Mint**

```
# 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 RedHat, Centos, or Fedora**

```
# /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 Debian, Ubuntu, or Linux Mint**

```
# /usr/bin/openssl genrsa -aes128 2048 > /etc/ssl/private/server.key
```

3. Create a new self-signed SSL certificate

**On RedHat, Centos, or Fedora**

```
# /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 Debian, Ubuntu, or Linux Mint**

```
# /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 Debian, Ubuntu, or Linux Mint**

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 `/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 `/etc/apache2/mods-enabled/ssl.conf`.



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 `/etc/sysconfig/apache2` and add the string `SSL` to the variable `APACHE_SERVER_FLAGS` so it looks like this:

```
APACHE_SERVER_FLAGS="SSL"
```



On RedHat, Centos, or Fedora

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.

```
# systemctl restart httpd
```

or

```
# systemctl restart apache2
```

✍ Exercise 7.7: Create a Certificate Signing Request

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

✓ Solution 7.7

1. Create a new private key



On RedHat, Centos, or Fedora

```
# /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 Debian, Ubuntu, or Linux Mint

```
# /usr/bin/openssl genrsa -aes128 2048 > /etc/ssl/private/server.key
```

2. Create a new CSR.

**On RedHat, Centos, or Fedora**

```
# /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 Debian, Ubuntu, or Linux Mint**

```
# /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/`):



`/new-cgi/foo.cgi`

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

A copy of this file (`foo.cgi`) can be found in the tarball in the `LFS211/SOLUTIONS/s_08/` directory.

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.



`newscripts.conf`

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

A copy of this file (`newscripts.conf`) can be found in the tarball in the `LFS211/SOLUTIONS/s_08/` directory.

**On RedHat, Centos, or Fedora**

```
/etc/httpd/conf.d/newscripts.conf
```

**On Debian, Ubuntu, or Linux Mint**

```
/etc/apache2/sites-enabled/newscripts.conf
```

**On openSUSE**

```
/etc/apache2/conf.d/newscripts.conf
```

3. If required, enable the cgi module:

**On Debian, Ubuntu, or Linux Mint**

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**Please Note**

If you have done the virtualhost lab from the previous chapter you need to add these two lines to `namevhost.conf` in the `_default_namevhost` section:

```
RewriteEngine on
RewriteOptions inherit
```

Make any URIs which begin 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:

**rewrite.conf**

```
RewriteEngine on
RewriteRule ^/foo/(.*) /scripts/foo.cgi?$1 [L,PT]
```

**On RedHat, Centos, or Fedora**

```
/etc/httpd/conf.d/rewrite.conf
```

**On Debian, Ubuntu, or Linux Mint**

```
/etc/apache2/sites-enabled/rewrite.conf
```

**Please Note**

On **Ubuntu** and **Debian** systems the **rewrite** commands seem to work best in: `/etc/apache2/sites-enabled/000-default.conf` file. If using **Ubuntu** or **Debian** systems, put the following **inside** the last **virtualhost** stanza of `000-default.conf`:

```
RewriteEngine on
RewriteOptions inherit
RewriteRule ^/foo/(.*) /scripts/foo.cgi?$1 [L,PT]
```

**On openSUSE**

```
/etc/apache2/conf.d/rewrite.conf
```

The documentation for the rewrite flags can be found at: <https://httpd.apache.org/docs/2.4/rewrite/flags.html>

2. If required, enable the rewrite module:**On openSUSE**

Edit `/etc/sysconfig/apache2` and edit the line with the `APACHE_MODULES`, and add the value **rewrite**:

**On Debian, Ubuntu, or Linux Mint**

Enable the rewrite module to be loaded

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

or

```
# a2enmod rewrite
```

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`:**

**status.conf**

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

**On RedHat, Centos, or Fedora**

```
/etc/httpd/conf.d/status.conf
```

**On Debian, Ubuntu, or Linux Mint**

```
/etc/apache2/mods-available/status.conf
```

**On openSUSE**

```
/etc/apache2/conf.d/status.conf
```

2. Confirm or create the **server-status** directory exists in your distributions **DOCUMENTROOT**.

**On RedHat, Centos, or Fedora**

```
/var/www/html/server-status/
```

**On Debian, Ubuntu, or Linux Mint**

```
/var/www/html/server-status/
```

**On openSUSE**

```
/srv/www/htdocs/server-status
```

3. If required, enable the **status** module:

**On openSUSE**

Edit `/etc/sysconfig/apache2` and edit the line with the `APACHE_MODULES`, and add the value `status`:

**On Debian, Ubuntu, or Linux Mint**

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 in the file locations listed below:



index.html

```
<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 RedHat, Centos, or Fedora

`/var/www/html/magic/index.html`



On openSUSE

`/srv/www/htdocs/magic/index.html`



On Debian, Ubuntu, or Linux Mint

`/var/www/html/magic/index.html`

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



foo.html

this is the foo include



On RedHat, Centos, or Fedora

`/var/www/html/includes/foo.html`

**On openSUSE**

```
/srv/www/htdocs/includes/foo.html
```

**On Debian, Ubuntu, or Linux Mint**

```
/var/www/html/magic/includes/foo.html
```

**bar.html**

```
this is the bar include
```

**On RedHat, Centos, or Fedora**

```
/var/www/html/includes/bar.html
```

**On openSUSE**

```
/srv/www/htdocs/includes/bar.html
```

**On Debian, Ubuntu, or Linux Mint**

```
/var/www/html/magic/includes/bar.html
```

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

**magic.conf**

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

**On RedHat, Centos, or Fedora**

```
/etc/httpd/conf.d/magic.conf
```

**On openSUSE**

```
/etc/apache2/conf.d/magic.conf
```

**On Debian, Ubuntu, or Linux Mint**

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

If required, enable the **include** module:



On Debian, Ubuntu, or Linux Mint

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

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

4. 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. Using your appropriate installer, ensure **Postfix** is installed:

```
# yum|zypper|apt install postfix
```



Please Note

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
```

**Please Note**

Be aware the firewall may interfere with this test.

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

**Please 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 neat
.
quit
```

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

**Please 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 RedHat, Centos, or Fedora**

```
# yum install dovecot mutt
```

**On openSUSE**

```
# zypper install dovecot mutt
```



On Debian, Ubuntu, or Linux Mint

```
# 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 `/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 `/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 `/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>/
```



Please Note

You may have to connect twice with **mutt** to verify the **imap** server is working.

The server may already be set up for SSL/StartTLS with a dummy SSL certificate.

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 `/etc/dovecot/conf.d/10-ssl.conf` and add or edit these line:

```
ssl = required
```

- On **OpenSUSE**:

- (a) Add or edit `/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 `/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 an incorrectly set up **DNS** server, or enforced **ssl**, use this setting for your lab as well:

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



Very Important

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



Please 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 `/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, \
    reject"
```

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**.**Please Note**

Be advised that any system listed in **permit_mynetworks** will be allowed to relay.

There are other **Postfix** options to control relaying of mail other than **smtpd_recipient_restrictions** that may be more suitable for a live installations. This example shows the restrictions being stacked and applied to the **smtpd_recipient_restrictions** control element.

Feel free to experiment with others such as: **smtpd_relay_restrictions**.

The current settings of **permit_mynetworks** in conjunction with **mynetworks_style** will allow the local system to relay without authentication.

If you wish to test authentication on a single machine eliminate the **permit_mynetworks** entry from **smtpd_recipient_restrictions** to force all systems attempting to relay to authenticate.

**Please Note**

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

```
$ 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 **Ubuntu**:


```
# /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:
Note: CentOS and Ubuntu have different key locations, only Ubuntu shown.


```
# 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:

**Please Note**

You may have to do this twice to get the key data.
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
```

**Please Note**

There is no option for AUTH until after you start the TLS session.
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 localhost interface.

Solution 10.1

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

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

Solution 10.2



Please Note

Notice the `.` at the end of the destination of the **rsync** command.

```
$ rsync -av /tmp/transfer-lab/*.bin root@localhost:/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 RedHat, Centos, or Fedora

```
# mkdir -m 730 /var/ftp/uploads/
# chown root.ftp /var/ftp/uploads/
```



On Debian, Ubuntu, or Linux Mint

```
# mkdir -m 730 /srv/ftp/uploads/
# chown root.ftp /srv/ftp/uploads/
```



On openSUSE

```
# mkdir -m 730 /srv/ftp/uploads/
# chown root.ftp /srv/ftp/uploads/
```

2. Edit `vsftpd.conf` and enable anonymous uploads. Add the following options:



`/etc/vsftpd/vsftpd.conf` or `/etc/vsftpd.conf`

```
anon_upload_enable=yes
anonymous_enable=yes
```



Please Note

On **Ubuntu** or **OpenSUSE** you must also change the option **write_enable** to match:

```
write_enable=YES
```

3. Test the ftp anonymous upload works.

✍ 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 `/etc/rsyncd.conf` and add these contents:

**/etc/rsyncd.conf**

```
[default]
path      = /srv/rsync
comment   = default rsync files
```

**Please Note**

On **OpenSUSE** you must also comment out or remove the line:
`hosts allow = trusted.hosts`

2. Enable the **rsyncd** daemon.

```
# systemctl start rsyncd
# systemctl enable rsyncd
```

3. To see the rsync modules that are shared:

```
$ rsync localhost::
```

```
1 default          default rsync files
```


Chapter 11

Advanced Networking



11.1 Labs

Exercise 11.1: Create a VLAN-trunked interface



Please 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:



Enable VLANs

Edit `/etc/sysconfig/network` and add the following content:



`/etc/sysconfig/network`

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

2. Create the interface file:

**Create a VLAN interface configuration file:**

On **CentOS**, edit or create `/etc/sysconfig/network-scripts/ifcfg-<INTERFACE>.7` with the following content:



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

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

A copy of this file (`ifcfg-<INTERFACE>.7-redhat`) can be found in the tarball in the [LFS211/SOLUTIONS/s_11/](#) directory.

**On openSUSE Create the interface file**

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

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

A copy of this file (`ifcfg-<INTERFACE>.7-opensuse`) can be found in the tarball in the [LFS211/SOLUTIONS/s_11/](#) directory.

**Create the interface file**

`/etc/network/interfaces`

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

A copy of this file (`interfaces`) can be found in the tarball in the [LFS211/SOLUTIONS/s_11/](#) directory.
The package `vlan` may need to be installed and the kernel-module `8021q` loaded.

3. Start the newly defined interface:

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

4. As an optional step, the `ip` command can create temporary vlan interfaces for testing. In this case the interface is called `eth0`, the vlan id is 8. Check the `/proc/net/vlan` directory for configuration and usage information:

```
[root@centos ~]# ip link
```

```

1 1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue state UNKNOWN mode DEFAULT group default \
2   qlen 1000 link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
3 2: eth0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc pfifo_fast state UP mode DEFAULT group default \
4   qlen 1000 link/ether 52:54:00:ab:3b:11 brd ff:ff:ff:ff:ff:ff

```

```
[root@centos ~]# ip link add link eth0 name eth0.v8 type vlan id 8
```

```
[root@centos ~]# ip -d link show eth0.v8
```

```

1 4: eth0.v8@eth0: <BROADCAST,MULTICAST> mtu 1500 qdisc noop state DOWN mode DEFAULT group default\
2   qlen 1000 link/ether 52:54:00:ab:3b:11 brd ff:ff:ff:ff:ff:ff promiscuity 0
3   vlan protocol 802.1Q id 8 <REORDER_HDR> addrngenmode eui64 numtxqueues 1 numrxqueues \
4   gso_max_size 65536 gso_max_segs 65535

```

```
[root@centos ~]# cat /proc/net/vlan/eth0.v8
```

```

1 eth0.v8  VID: 8          REORDER_HDR: 1  dev->priv_flags: 1
2          total frames received              0
3          total bytes received              0
4          Broadcast/Multicast Rcvd          0
5
6          total frames transmitted          0
7          total bytes transmitted          0
8 Device: eth0
9 INGRESS priority mappings: 0:0  1:0  2:0  3:0  4:0  5:0  6:0  7:0
10 EGRESS priority mappings:

```

```
[root@centos ~]# cat /proc/net/vlan/config
```

```

VLAN Dev name      | VLAN ID
Name-Type: VLAN_NAME_TYPE_RAW_PLUS_VID_NO_PAD
eth0.v8           | 8   | eth0

```

```
[root@centos ~]# ip addr add 192.168.42.100/24 dev eth0.v8
```

```
[root@centos ~]# ip link set dev eth0.v8 up
```

```
[root@centos ~]# ip -d addr show eth0.v8
```

```

1 4: eth0.v8@eth0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc noqueue state UP group default \
2   qlen 1000 link/ether 52:54:00:ab:3b:11 brd ff:ff:ff:ff:ff:ff promiscuity 0
3   vlan protocol 802.1Q id 8 <REORDER_HDR> numtxqueues 1 numrxqueues 1 \
4   gso_max_size 65536 gso_max_segs 65535
5   inet 192.168.42.100/24 scope global eth0.v8  valid_lft forever preferred_lft forever
6   inet6 fe80::5054:ff:feab:3b11/64 scope link valid_lft forever preferred_lft forever

```

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:



create or edit the static route file



/etc/sysconfig/network-scripts/route-INTERFACE

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 static route file



/etc/sysconfig/network/ifroute-INTERFACE

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>
```



update the static route information



/etc/network/interfaces

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 using your favorite installer:

```
# yum install ntp
# zypper install ntp
# apt-get install ntp
```

2. Configure the NTP server to query the NTP pool and allow for anonymous traffic.

Edit `/etc/ntp.conf` and change the `server X.X.X.X` lines to match these:



`/etc/ntp.conf`

```
server 0.pool.ntp.org
server 1.pool.ntp.org
server 2.pool.ntp.org
server 3.pool.ntp.org
```

3. Restart **ntp** server with the appropriate command:

```
# systemctl restart ntpd
# systemctl restart ntp
```

4. Query your new timeserver:

```
$ ntpq -p
```



Please 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 with your appropriate installer:

```
# yum | zypper | apt install squid
```

2. This exercise will be accessing web sites on the Internet. Verify the default route for your system.

```
# ip route
```

Verify and record your IP address on the network with the default route.

```
# ip addr
```

3. Create an ACL for your network. Edit `/etc/squid/squid.conf` and add an **ACL** above the existing **ACL**'s.



squid.conf: Network ACL

```
acl examplenetwork src 192.168.122.0/24
```

4. Locate the following line in the **squid.conf** file:

**squid.conf**

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

5. Explicitly allow HTTP access for the newly created ACL, by adding this line below the line above:

**squid.conf allow access**

```
http_access allow examplenetwork
```

6. Test the syntax of **squid.conf**:

```
# squid -k parse
```

7. Start or restart the Squid daemon:

```
# systemctl restart squid
```

8. Test the proxy:

- Configure a web browser to use your new proxy.
- Visit a known good URI (<http://www.foxnews.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://www.cbc.com/> using Squid

Connections made to the Canadian news service **CBC** using <http://www.cbc.com> brings up a page that is not the news service.

Block <http://www.cbc.com> and allow <http://www.cbc.ca>.

Solution 12.2

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

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

**squid.conf site restriction**

```
acl blockedsite url_regex ^http://.*cbc.com/.*$
http_access deny blockedsite
```

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

```
# squid -k reconfigure
```

3. Test the ACL by trying to browse to <http://www.cbc.com>

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 RedHat, Centos, or Fedora

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



On openSUSE

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



On Debian, Ubuntu, or Linux Mint

```
# 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 `/etc/exports` and add the following content:



`/etc/exports`

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

2. Start or restart the NFS service:



On RedHat, Centos, or Fedora

```
# systemctl restart nfs
```



On openSUSE

```
# systemctl restart nfsserver
```



On Debian, Ubuntu, or Linux Mint

```
# systemctl restart nfs-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 that the **NFS** server is installed:

**On RedHat, Centos, or Fedora**

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

**On openSUSE**

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

**On Debian, Ubuntu, or Linux Mint**

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

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

**smb.conf**

```
[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
```

A copy of this file (`smb.conf`) can be found in the tarball in the [LFS211/SOLUTIONS/s_13/](#) directory.

3. Start or restart the samba service:

**On RedHat, Centos, or Fedora**

```
# systemctl restart smb
```

**On openSUSE**

```
# systemctl restart
```

**On Debian, Ubuntu, or Linux Mint**

```
# systemctl restart smbd
```

4. Check if the share is available. The password is not required; when prompted press Enter to continue as **anonymous**.

```
$ smbclient -L 127.0.0.1
```

5. From another machine or (`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 the 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 `/etc/samba/smb.conf` and add this stanza to the bottom.



private stanza in smb.conf

```
[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 `/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 `/etc/fstab` for the CIFS mount:

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

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

```
# systemctl daemon-reload
```

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

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

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

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

Using **systemd.automount**, 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 `/etc/fstab` as shown.



`/etc/fstab`

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

2. Instruct **systemd** to re-read `/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.

Alternate (and optional) solution using **systemd** without the `fstab`.

1. Disable or remove the mount entry just created from the `/etc/fstab`.
2. Create the two files listed below:

The first file is: `/etc/systemd/system/home-share-nfs.mount`



/etc/systemd/system/home-share-nfs.mount

```
# store in /etc/systemd/system/home-share-nfs.mount
# unit name must match the mount point (the where matches the unit file name)

[Unit]
Description=basic mount options file for auto-mounting home-share-nfs

[Mount]
What=localhost:/home/export/nfs
Where=/home/share/nfs
Type=nfs

[Install]
WantedBy=multi-user.target
```

A copy of this file (`home-share-nfs.mount`) can be found in the tarball in the [LFS211/SOLUTIONS/s_13/](#) directory. The second file is:



/etc/systemd/system/home-share-nfs.automount

```
# store in /etc/systemd/system/home-share-nfs.automount
# unit name must match the mount point (the where matches the unit file name)

[Unit]
Description=options for our home-share-nfs mount

[Automount]
Where=/home/share/nfs
TimeoutIdleSec=10

[Install]
WantedBy=multi-user.target
```

A copy of this file (`home-share-nfs.automount`) can be found in the tarball in the [LFS211/SOLUTIONS/s_13/](#) directory. There are restrictions on the file names; consult the **man** page for **automount** for additional details.

3. Tell **systemd** about the new configuration files:

```
# systemctl daemon-reload
```

4. Confirm the mount unit file is disabled and stopped:

```
# systemctl disable --now home-share-nfs.service
```

5. Verify or activate the automount unit file:

```
# systemctl enable --now home-share-nfs.automount
```

6. Test the automount. As before, after about 10 seconds the auto-mounted device should unmount.

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

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: Exploring iptables firewalls



Very Important

This exercise expects to be run on a “class VM” system where we can freely disable and change the firewall rules. If you are running on a machine that requires “ssh login”, **BE VERY CAREFUL**: you may lock yourself out.

- Listen to port 4200 and display any text coming in.
- Log any new connections to the system log.
- Log any established connections to the system log.
- Using **iptables**, block all established connections to port 4200.
- Save, flush and restore the firewall

Solution 15.1

1. Open port 4200 and display any text.
2. Using your appropriate installer, verify **netcat** is installed:

```
# yum|zypper|apt install netcat-openbsd|ncat  
# yum|zypper|apt install telnet
```
3. Open three terminal sessions: one root shell and two user (student) shells.
4. In the root shell, list and delete any iptables rules.

```
# iptables -vnL  
# iptables -F
```

5. In one student shell, open a listener on port 4200 with **netcat**:

```
$ netcat -4 -l 4200
```

6. In the other student window, use **telnet** to connect to localhost port 4200:

```
$ telnet localhost 4200
```

The telnet session should connect and appear hung, as it is waiting for input: type something. The typed characters should appear on the **netcat** terminal window. After verifying the **netcat** connection, stop **netcat** with CTRL-C.

7. Log all new connections with iptables. From the root shell terminal:

```
# iptables -A INPUT -p tcp -m tcp -m state --state NEW -j LOG --log-level info --log-prefix "LF NEW "
```

8. Monitor new connections via the system log. On the root terminal session:

```
# journalctl -f
```

Stop and restart the **netcat** and **telnet** commands while watching the **journalctl** output; you should see the connection packet header with the text **LF NEW** visible.

9. Add another rule to the firewall to log established connections:

```
# iptables -A INPUT -p tcp -m tcp -m state --state ESTABLISHED -j LOG \
    --log-level info --log-prefix "LF ESTABLISHED"
```

10. Test and observe the output in the root **journalctl** log. The log should show the initial connection and the packet headers of all the transactions between telnet and netcat.

11. Add an iptables rule to reject new connections on port 4200. If there is an established connection, it should continue to work.

```
# iptables -A INPUT -p tcp -m tcp --dport 4200 -m state --state ESTABLISHED -j REJECT
```

If there was an established session, it should continue to function. Any new connections should fail.

12. If we examine the log output, we see the ESTABLISHED headers being logged, but we cannot connect. This is due to the order of the rules. To change the existing iptables records, first save the current configuration:

```
# iptables-save > /tmp/ip.save
```

With your favorite editor, modify **/tmp/ip.save** and **move** the line that ends in REJECT to be before the ESTABLISHED line:



/tmp/ip.save Edited

```
root@ubuntu:~# cat /tmp/ip.save
# Generated by iptables-save v1.6.1 on Sat Jul 27 09:59:20 2019
*filter
:INPUT ACCEPT [0:0]
:FORWARD ACCEPT [0:0]
:OUTPUT ACCEPT [0:0]
-A INPUT -p tcp -m tcp -m state --state NEW -j LOG --log-prefix "New Connection" --log-level 6
-A INPUT -p tcp -m tcp --dport 4200 -m state --state ESTABLISHED -j REJECT --reject-with
↳ icmp-port-unreachable
-A INPUT -p tcp -m tcp -m state --state ESTABLISHED -j LOG --log-prefix "LFT ESTABLISHED "
↳ --log-level 6
COMMIT
# Completed on Sat Jul 27 09:59:20 2019
```

13. Clear out existing iptables rules:

```
# iptables -F
```

14. Restore the modified configuration file:

```
# iptables-restore /tmp/ip.save
```

Re-test as desired, there should not be any more ESTABLISHED messages for destination port 4200.

15. Flush out the iptables records to prepare for the next exercise:

```
# iptables -F
```

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 RedHat, Centos, or Fedora

```
# service iptables save
```



On openSUSE

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
```

✍ Exercise 15.3: Using nftables, Enable a firewall which blocks all unwanted traffic

This is a repeat of the last exercise using **nftables**.

- 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 15.3

1. Clear out any iptables rules:

```
# iptables -F
```

2. Also flush out any **nftables** rules and constructs

```
# nft flush ruleset
```

3. Add a table to **nft**, remember there are no pre-configured tables in **nft** by default. The table will be for both IPv4 and IPv6.

```
# nft add table inet filter_it
```

4. Add a **chain** to the table and declare which **hook** will be used in the TCP stack:

```
# nft add chain inet filter_it input_stuff {type filter hook input priority 0 \; }
```

5. Add in the rule to accept port 22 (ssh) traffic:

```
# nft add rule inet filter_it input_stuff tcp dport 22 accept
```

6. Look at the firewall constructs so far:

```
# nft list table inet filter_it
```

7. Add connection tracking for established and related packs:

```
# nft add rule inet filter_it input_stuff ct state established,related accept
```

8. Lets add a packet counter:

```
# nft add rule inet filter_it input_stuff counter
```

9. We should listen to all traffic on the loopback adapter

```
# nft add rule inet filter_it input_stuff iifname lo accept
```


10. Set the default to ignore everything:

```
# nft add rule inet filter_it input_stuff drop
```

11. Examining the listing of our firewall shows the rules in a poorly organized fashion:


```
# nft list table inet filter_it

1 table inet filter_it {
2     chain input_stuff {
3         type filter hook input priority 0; policy accept;
4         tcp dport ssh accept
5         ct state established,related accept
6         counter packets 2 bytes 140
7         iifname "lo" accept
8         drop
9     }
10 }
```

12. We can use the output of `nft list table filter_it` as input to **nft**. First, save the configuration. If required create the directory:

```
# mkdir /etc/nftables
# nft list table inet filter_it > /etc/nftables/00-filter_it.nft
```

13. Arrange the rules in a more logical sequence



```
/etc/nftables/00-filter_it.nft

table inet filter_it {
    chain input_stuff {
        type filter hook input priority 0; policy accept;
        ct state established,related accept
        iifname "lo" accept
        tcp dport ssh accept
        counter packets 0 bytes 0
        drop
    }
}
```

14. Test the new configuration:

```
# nft flush ruleset
# nft -f /etc/nftables/00-filter_it.nft
# nft list table inet filter_it
```

15. Add the nftables configuration to the startup sequence and reboot to test:



```
On RedHat, Centos, or Fedora

# systemctl disable firewalld
# systemctl enable nftables
# echo 'include "/etc/nftables/00-filter_it.nft"' >> /etc/sysconfig/nftables.conf
```



```
On Debian, Ubuntu, or Linux Mint
```

```
# mkdir /etc/nftables
# systemctl disable firewalld
```



```
# systemctl enable nftables
# echo 'include "/etc/nftables/00-filter_it.nft"' >> /etc/nftables.conf
```

OpenSUSE needs extra help, all files referenced here are in the SOLUTIONS directory.



On openSUSE

```
# mkdir /etc/nftables
# cp SOLUTIONS/nftables.service /etc/systemd/system/nftables.service
# echo 'include "/etc/nftables/00-filter_it.nft"' >> /etc/nftables.conf
# systemctl daemon-reload
# systemctl disable firewalld
# systemctl enable nftables
```

Chapter 16

LXC Virtualization Overview



16.1 Labs

Exercise 16.1: Install and test web server as a LXC application

Install, run and test the **LXC** container and a **web server**.

LXC 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 Linux Containers as a non-root user, but in this exercise we will execute the **LXC** commands as root. The configuration occurs on the Virtual Machine unless otherwise noted.

Solution 16.1

1. Make sure **LXC** is installed.

- On **CentOS8**:

```
# dnf install lxc lxc-templates lxcfs lxc-doc lxc-libs
```
- On **Ubuntu**:

```
# apt-get install lxc
```
- On **CentOS7** the command **lxc-net** is not available.



CentOS7 ONLY! Using libvirtd networking with LXC

The command **lxc-net** is not available on **CentOS7**, however, **libvirt** is. The steps to enable libvirt to provide networking for the lxc containers is here:

(a) Ensure libvirt is installed and running

```
# yum install libvirt\*
```

(b) Examine or correct the default config file for lxc is pointing to the libvirt bridge.

```
# cat /etc/lxc/default.conf
```

The names may vary slightly.



```
1 lxc.network.link = virbr0
2 lxc.network.type = veth
3 lxc.network.flags = up
```

- (c) Create or modify the `/etc/sysconfig/lxc-net` file.

```
# cat /etc/sysconfig/lxc-net
1 USE_LXC_BRIDGE="true"
2 LXC_BRIDGE="virbr0"
```

- (d) The devices may be **unmanaged** or **disconnected**, this is likely due to NetworkManager excluding the interfaces. If required add the virtual network interfaces to NetworkManager by creating a drop-in file.

```
# cat /etc/NetworkManager/conf.d/10-managed-devices-override.conf
1 [device]
2 match-device=interface-name:veth*,virbr0-nic
3 managed=1
4
```

These changes are usually all that is required. The lxc container should connect to the libvirt bridge.

If there is an address conflict libvirt will stop with a non-specific error. Carefully use the **virsh net-edit default** command to change the address for the bridge and the dhcp server.

CentOS7 skip forward in the instruction to the point "CentOS7 rejoins the lab"

2. Stop the firewall if is running on the Virtual-Machine. Best practice is to have a firewall in place between the Virtual-Machine and the LXC container, however, we will disable the firewall to stay focused on LXC-container for now.

```
# systemctl disable --now firewalld
```

3. Confirm or create the lxc-net configuration.



`/etc/sysconfig/lxc-net`

```
[root@localhost ~]# cat /etc/sysconfig/lxc-net
```

```
1 USE_LXC_BRIDGE="true"
2 LXC_BRIDGE="lxcbr0"
3 LXC_ADDR="10.0.3.1"
4 LXC_NETMASK="255.255.255.0"
5 LXC_NETWORK="10.0.3.0/24"
6 LXC_DHCP_RANGE="10.0.3.2,10.0.3.254"
7 LXC_DHCP_MAX="253"
8 LXC_DHCP_CONFILE=""
9 LXC_DOMAIN=""
```

4. Start the LXC network service and make it start on initialization automatically.

```
# systemctl enable --now lxc-net
```

5. Verify the new virtual adapter is present.

```
# ip addr
```

The test system's new adapter as viewed by `ip addr`

```
1 1xcbr0: <NO-CARRIER,BROADCAST,MULTICAST,UP> mtu 1500 qdisc noqueue state DOWN group default qlen 1000
2     link/ether 00:16:3e:00:00:00 brd ff:ff:ff:ff:ff:ff
3     inet 10.0.3.1/24 scope global 1xcbr0
4         valid_lft forever preferred_lft forever
```

6. **CentOS7** rejoins the lab here.

7. Create the LXC container.

```
# lxc-create -n bucket -t download
1 [root@localhost ~]# lxc-create -n bucket -t download
2 Setting up the GPG keyring
3 Downloading the image index
4
5 ---
6 DIST      RELEASE      ARCH      VARIANT      BUILD
7 ---
8 ..... snip ...
9 centos      8-Stream      amd64      default      20200401_07:08
10 centos      8-Stream      arm64      default      20200401_07:08
11 centos      8-Stream      ppc64el    default      20200401_07:08
12 centos      8             amd64      default      20200401_07:39
13 centos      8             arm64      default      20200401_07:39
14 centos      8             ppc64el    default      20200401_07:39
15 debian      bullseye      amd64      default      20200401_05:24
16 debian      bullseye      arm64      default      20200401_05:24
17 debian      bullseye      armel      default      20200401_05:24
18 debian      bullseye      armhf      default      20200401_05:54
19
20 ..... snip ....
21
22 Distribution:
```

centos

```
1 Release:
```

8

```
1 Architecture:
```

amd64

```
1 Downloading the image index
2 Downloading the rootfs
3 Downloading the metadata
4 The image cache is now ready
5 Unpacking the rootfs
6
7 ---
8 You just created a Centos 8 x86_64 (20200401_07:39) container.
```

It is possible to have a command line option with the options specified on the command line.



command line `lxc-create`

```
lxc-create -n bucket1 -t download -- -d centos -r 8 -a amd64
```

8. Start the container.



Please Note

CentOS 7 requires an additional option "-d" to run in background. The defaults have been updated in later releases.

```
# lxc-start -n bucket
```

9. List the installed containers.

```
# lxc-ls -f
```

10. Open a new terminal session and attach to the container.

```
# lxc-attach -n bucket
```

11. Explore the LXC container, verify the network address and functionality.

12. In the VM window, not the container, create and start two more LXC containers called **bucket1** and **bucket2**

13. One could attach to each container to add additional software like a web-server or use the **lxc-attach** command to run commands inside the containers.

Add the **nginx** web server to all three containers.

```
# lxc-attach -n bucket -- /bin/dnf install -y nginx
# lxc-attach -n bucket1 -- /bin/dnf install -y nginx
# lxc-attach -n bucket2 -- /bin/dnf install -y nginx
```

14. `lxc-attach -n bucket -- /bin/systemctl start nginx`

15. Verify each of the containers web server is responding. Use **lxc-ls -f** to obtain a list of the IP addresses.

16. It may be helpful to use a text based browser, OPTIONALLY install or compile lynx.

```
git clone https://github.com/nabetaro/lynx.git
./autogen.sh
./configure
make
sudo make install
```

17. Create a new web page different on each server. Something like **Hello World from bucket1** and test.

```
# lxc-attach bucket2
# echo "Hello World from bucket2" > /usr/share/nginx/html/index.html
# exit
# lynx -dump http://10.3.0.200
```

18. Using **name virtual hosts** requires the web server name is resolvable. Add the hostname **bucket** to the `/etc/hosts` on the Virtual-Machine and the container **bucket**. Be mindful of duplicate names with loopback addresses.

19. Enable load balancing in a round-robin configuration.



/etc/nginx/conf.d/00-load.conf

```
upstream bucket {
    server 10.0.3.174;      #ip for bucket1
    server 10.0.3.200;      #ip for bucket2
}

server {
    listen 80;
```



```
server_name bucket;
location / {
    proxy_pass http://bucket;
}
```

20. Restart nginx on the **bucket** container.

```
# systemctl restart nginx
```

21. Test the round robin balancer is working. On the VM open the default web page on the **bucket** container, if everything is fine a message should be returned from either **bucket1** or **bucket2**. The output should look similar to:

```
[root@localhost ~]# /usr/local/bin/lynx -dump http://bucket
```

```
1 Hello World from bucket1
```

```
[root@localhost ~]# /usr/local/bin/lynx -dump http://bucket
```

```
1 Hello World from bucket2
```

Exercise 16.2: Duplicate a LXC container.

Should the need to increase the number of servers available arise, **lxc-copy** can simplify the process.



Please Note

Note that older versions of **LXC** have a **lxc-clone** command and newer versions have the **lxc-copy** command. Use the man pages to see the difference.

Add another exact copy of the web server containers (not the load balancer) and have all of the containers and web servers automatically start at boot time.

Solution 16.2

1. Verify the LXC container **bucket2** is stopped, if it is running stop it.

```
# lxc-ls -f
```

2. Duplicate the LXC container

```
# lxc-copy -n bucket2 -N bucket3
```

3. Edit the web page to reflect which container this is.

```
# lxc-attach bucket3
# echo "Hello World from bucket3" > /usr/share/nginx/html/index.html
# exit
```

4. From the **VM**, start the container **bucket3** and record its IP address.

```
# lxc-start -n bucket3
# lxc-ls -f
```

5. Edit the **nginx** configuration, `/etc/nginx/conf.d/00-load.conf` file and add in the new node, **bucket3**, to the load-balance list on the container **bucket**.

6. Re-load the nginx web server on **bucket**.

```
# systemctl reload nginx
```

Start the nginx server on **bucket3**.

```
# systemctl --now enable nginx
```

Test to ensure all load balance members are working.

```
# /usr/local/bin/lynx -dump http://bucket
```


Chapter 17

High Availability



17.1 Labs

There is no lab to complete for this chapter.

Chapter 18

Database



18.1 Labs

Exercise 18.1: Install and test MariaDB

This exercise will install a **MariaDB** relational database engine and then test that the **RDBMS** is working by adding a couple of tables to a new database running as the user `student`.

1. Make sure **MariaDB** is installed. Use whichever command is right for your distribution:

```
$ sudo yum install mariadb-server
$ sudo zypper install mariadb-server
$ sudo apt-get install mariadb-server
```

2. Start the **mariadb** service.:

```
$ sudo systemctl start mariadb
```

3. Make sure the service started successfully:

```
$ sudo systemctl status mariadb
```

4. Set the **mariadb** service to start on reboot:

```
$ sudo systemctl enable mariadb
```

5. **MariaDB** includes a script ([mysql_secure_installation](#)) for improving the security of the newly installed server. The script prompts the actions that it is about to implement. Some of the security items are:

- Set the password for the `root` accounts.
- Remove `root` accounts that are accessible from other than the `localhost` interface.
- Remove `anonymous-user` accounts.
- Remove the `test` database.
- and others.

There is currently no `root` account password in **MariaDB** so we will set it to **password** when prompted. Take the defaults for all prompts **except** `Remove test database`, we will be using the test database.

Here are the results of running the script:

```
[student@CentOS7 ~]$ mysql_secure_installation
```

```
NOTE: RUNNING ALL PARTS OF THIS SCRIPT IS RECOMMENDED FOR ALL MariaDB
SERVERS IN PRODUCTION USE! PLEASE READ EACH STEP CAREFULLY!
```

```
In order to log into MariaDB to secure it, we'll need the current
password for the root user. If you've just installed MariaDB, and
you haven't set the root password yet, the password will be blank,
so you should just press enter here.
```

```
Enter current password for root (enter for none):
OK, successfully used password, moving on...
```

```
Setting the root password ensures that nobody can log into the MariaDB
root user without the proper authorization.
```

```
Set root password? [Y/n] y
New password:
Re-enter new password:
Password updated successfully!
Reloading privilege tables..
... Success!
```

```
By default, a MariaDB installation has an anonymous user, allowing anyone
to log into MariaDB without having to have a user account created for
them. This is intended only for testing, and to make the installation
go a bit smoother. You should remove them before moving into a
production environment.
```

```
Remove anonymous users? [Y/n] y
... Success!
```

```
Normally, root should only be allowed to connect from 'localhost'. This
ensures that someone cannot guess at the root password from the network.
```

```
Disallow root login remotely? [Y/n] y
... Success!
```

```
By default, MariaDB comes with a database named 'test' that anyone can
access. This is also intended only for testing, and should be removed
before moving into a production environment.
```

```
Remove test database and access to it? [Y/n] n
... skipping.
```

```
Reloading the privilege tables will ensure that all changes made so far
will take effect immediately.
```

```
Reload privilege tables now? [Y/n] y
... Success!
```

```
Cleaning up...
```

```
All done! If you've completed all of the above steps, your MariaDB
installation should now be secure.
```

```
Thanks for using MariaDB!
[student@CentOS7 ~]$
```

6. Verify the **MariaDB** server is ready with the **mysqladmin** command:

```
[student@CentOS7 ~]$ mysqladmin -u root -p version
```

Enter password:

```
mysqladmin Ver 9.0 Distrib 5.5.56-MariaDB, for Linux on x86_64
Copyright (c) 2000, 2017, Oracle, MariaDB Corporation Ab and others.
```

```
Server version 5.5.56-MariaDB
Protocol version 10
Connection Localhost via UNIX socket
UNIX socket /var/lib/mysql/mysql.sock
Uptime: 2 hours 46 min 4 sec
```

```
Threads: 2 Questions: 52 Slow queries: 0 Opens: 15 Flush tables: 2
Open tables: 41 Queries per second avg: 0.005
```

7. Connect to the new database as the root user:

```
$ mysql -u root -p
```

8. Add the user student with the password password to the database. The user student should be able to log-in from anywhere:

```
MariaDB [(none)]> CREATE USER 'student'@'%' IDENTIFIED BY 'password';
```

9. Verify the user was created:

```
MariaDB [(none)]> SELECT USER, HOST FROM mysql.user;
```

10. Grant the new user "ALL" privileges to the database **test**:

```
MariaDB [(none)]> GRANT ALL PRIVILEGES ON test.* TO 'student'@'%';
```

11. Log out of the database as the user root and log back in as the user student:

```
MariaDB [(none)]> exit
$ mysql -u student -p test
```

Notice the prompt now has the database name **test** instead of **none**.

12. Add some tables to the database:



Adding tables

```
MariaDB [test]> CREATE TABLE Courses (
-> Courseno INT NOT NULL ,
-> Title VARCHAR(100) NOT NULL,
-> Description VARCHAR(200),
-> PRIMARY KEY ( Courseno )
-> );
Query OK, 0 rows affected (0.01 sec)

MariaDB [test]> CREATE TABLE Employees (
-> Empno SMALLINT ZEROFILL,
-> First VARCHAR(40),
-> Last VARCHAR(40),
-> PRIMARY KEY ( Empno ) );
Query OK, 0 rows affected (0.01 sec)

MariaDB [test]> CREATE TABLE Roster (
-> Uuid SMALLINT NOT NULL AUTO_INCREMENT,
```



```
-> CoursenoR INT NOT NULL,
-> EmpnoR SMALLINT ZEROFILL, PRIMARY KEY ( Uuid ) );
Query OK, 0 rows affected (0.00 sec)

MariaDB [test]>
```

13. Insert data into the tables



Inserting Data

```
MariaDB [test]> INSERT INTO Courses VALUES (1,'Rocket Design','Acme Rocket Basics');
Query OK, 1 row affected (0.00 sec)

MariaDB [test]> INSERT INTO Courses VALUES (2,'Solid Fuel','Acme Chemical Propulsion');
Query OK, 1 row affected (0.00 sec)

MariaDB [test]> INSERT INTO Courses VALUES (3,'Gantry Basics','Hold rocket up');
Query OK, 1 row affected (0.00 sec)

MariaDB [test]> INSERT INTO Employees (Empno,First,Last)
-> Values (02483,'Alice','Cooper'),
-> (10746,'Bob','Marley'),
-> (02318,'Cindy','Lauper');
Query OK, 3 rows affected (0.01 sec)
Records: 3 Duplicates: 0 Warnings: 0

MariaDB [test]>
```

14. Use the **SELECT** statement to query the tables.



Querying

```
MariaDB [test]> SELECT * FROM Courses;
+-----+-----+-----+
| Courseno | Title          | Description          |
+-----+-----+-----+
| 1 | Rocket Design | Acme Rocket Basics |
| 2 | Solid Fuel    | Acme Chemical Propulsion |
| 3 | Gantry Basics | Hold rocket up      |
+-----+-----+-----+
3 rows in set (0.00 sec)

MariaDB [test]> SELECT * FROM Employees ;
+-----+-----+-----+
| Empno | First | Last |
+-----+-----+-----+
| 02318 | Cindy | Lauper |
| 02483 | Alice | Cooper |
| 10746 | Bob   | Marley |
+-----+-----+-----+
3 rows in set (0.00 sec)
```

Chapter 19

System log



19.1 Labs

Exercise 19.1: Configure and test RSYSLOG Remote Logging

Overview

In this exercise the **rsyslog** daemon will be configured to receive messages from a different system. To facilitate this, network namespaces will be used to emulate a separate machine. Log messages will be generated in the namespace and sent to the logging server which will write the log messages to a file other than the default log message file.

1. Configure and test the network namespace.

(a) Add a network namespace called **netns1**:

```
# ip netns add netns1
```

(b) The **ip netns exec** command will allow program execution using the newly created network namespace. Verify the available network devices within the namespace:

```
# ip netns exec netns1 ip link list
```

(c) The **lo** interface is configured but is not up. Set up the **lo** interface:

```
# ip netns exec netns1 ip link set dev lo up
```

(d) Verify the **lo** interface is up and check its IP address:

```
# ip netns exec netns1 ip link list
# ip netns exec netns1 ip a
```

(e) The **lo** interface can be tested with a **ping** command, if it is executed within the namespace:

```
# ip netns exec netns1 ping -c3 127.0.0.1
```

(f) Create a pair of **virtual adapters** to be used to communicate between the namespace network stack and the default network stack.

Add the device pair:

```
# ip link add veth0 type veth peer name veth1
```

- (g) Connect one **virtual adapter** to the namespace:

```
# ip link set veth1 netns netns1
```

- (h) Assign the **virtual adapter** in the namespace an IP address:

```
# ip netns exec netns1 ifconfig veth1 10.1.1.1/24 up
```

- (i) Assign the other **virtual adapter** an address:

```
# ifconfig veth0 10.1.1.2/24 up
```

- (j) The adapters and their connection can be tested:

```
# ping -c3 10.1.1.1
```

- (k) Open a new terminal window and use **tcpdump** to monitor the traffic to and from the namespace adapter:

```
# tcpdump -i veth0 -n -B512
```

- (l) Experiment with the new network namespace.

2. Configure **rsyslog** for reception of syslog messages.

- (a) The input module for **TCP** or **UDP** reception has to be enabled in `/etc/rsyslog.conf`. Please be aware there are two formats for `rsyslog.conf`, as listed below:



rsyslog.conf: new format

```
# provides UDP syslog reception
module(load="imudp")
input(type="imudp" port="514")

# provides TCP syslog reception
module(load="imtcp")
input(type="imtcp" port="514")
```

A copy of this file (`rsyslog.conf-new`) can be found in the tarball in the `LFS211/SOLUTIONS/s_19/` directory.



rsyslog.conf: old format

```
# Provides UDP syslog reception
$ModLoad imudp
$UDPServerRun 514

# Provides TCP syslog reception
$ModLoad imtcp
$InputTCPServerRun 514
```

A copy of this file (`rsyslog.conf-old`) can be found in the tarball in the `LFS211/SOLUTIONS/s_19/` directory. Remove the comment marker `#` from both the **UDP** and **TCP** modules and run statements.

- (b) Restart the **rsyslog** daemon:

```
# systemctl restart rsyslog
```

- (c) Open a terminal window and monitor the messages being sent to your system log.



Please Note

Ubuntu uses `/var/log/syslog` and CentOS uses `/var/log/messages`.

```
# tail -f /var/log/syslog
```


There should be three terminal windows open:

- **tcpdump**
- A tail of system messages
- A **root shell** for issuing commands

(d) Test reception of syslog messages.

On the command window issue a **logger** message from the network namespace to the default system.

Multiple messages and rsyslog

If a message is sent multiple times, **rsyslog** collects and counts them indicating only that there is more of the same messages. To avoid this make each message unique.

There should be TCP or UDP traffic from 10.1.1.1 to 10.1.1.2 of type **syslog** on the **tcpdump** screen. The text should be visible on the system log monitoring screen:

```
# ip netns exec netns1 logger -n 10.1.1.2 from netns boo
```

The tcpdump window should show something like:

```
1
2 11:22:59.984357 IP 10.1.1.2.17500 > 10.1.1.255.17500: UDP, length 139
3 11:23:00.854595 IP 10.1.1.1.35329 > 10.1.1.2.514: SYSLOG user.notice, length: 127
```

and the system log monitor should have a message like:

```
1
2 Oct  8 11:25:56 yoga student from netns boo
```

3. The messages from the remote host will get logged in the same file as the servers own messages mixing them all together.

Use a **property filter** to isolate incoming log messages and direct them to a specified file.

(a) Add a property filter to the rsyslog receiver.

Open `/etc/rsyslog.d/00-property-filter.conf` as root and add the lines:



/etc/rsyslog.d/00-property-filter.conf

```
:FROMHOST-IP, isequal, "10.1.1.1" -/var/log/syslog-other
& ~
```

(b) Create the new log file and set the permissions the same as those of the existing log file.



log file permissions

Ubuntu and **CentOS** use different permissions, **Check your system**.

```
# touch /var/log/syslog-other
# chown syslog.adm /var/log/syslog-other
```

(c) Restart the rsyslog daemon to pick up the changes:

```
# systemctl restart rsyslog
```

(d) Test the property filter correctly logs the messages from the remote host to the server.

(e) Change the terminal monitoring the system log to now monitor the new log file:

```
# tail -f /var/log/syslog-other
```

(f) Send a system log message from the network namespace to the logging server and verify the message is written to the new log file:

```
# ip netns exec netns1 logger -n 10.1.1.2 Test to new remote log server.
```


Chapter 20

Package Management



20.1 Labs

Exercise 20.1: Building and installing from source, using git

In an earlier session we discussed **stress-ng** as a drop in replacement for the older **stress** utility; it 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>.

While your **Linux** distribution may have **stress-ng** within its packaging system, here 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:

```
1
2      .....
3 cc -O2 -Wall -Wextra -DVERSION="0.05.00" -O2 -c -o stress-key.o stress-key.c
4 stress-key.c:36:22: fatal error: keyutils.h: No such file or directory
5 #include <keyutils.h>
6 ^
7 compilation terminated.
8 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 the pre-packaged software supplied by 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
```



Please Note

While some program sources have a `make uninstall` target, not all do. Thus, uninstalling (removing, cleaning up) can be a pain, which is one reason packaging systems have been designed and implemented.

✍ Exercise 20.2: 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 should have already obtained 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
```

```
1 myappdebian-1.0/
2 myappdebian-1.0/Makefile
3 myappdebian-1.0/myhello.c
4 myappdebian-1.0/README
```

where we have:



README

Some very informative information should go in here :)

and



hello.c

```

1 #include <stdio.h>
2 #include <stdlib.h>
3 char hello_string[] = "hello world";
4 int main ()
5 {
6     printf ("\n%s\n\n", hello_string);
7     exit (EXIT_SUCCESS);
8 }

```

and



Makefile

```

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

all: $(TARGET)

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

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

clean:
            rm -f $(TARGET)

```



Please 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 `lab_makedeb.sh` which is also in the `LFS211/SOLUTIONS/s_20/` directory in the SOLUTIONS tarball.

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 20.3: Building an RPM

The `LFS211/SOLUTIONS/s_20/` directory in the contains an example of rpm creation. The relevant files are:

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