9 * Network traffic analysis

This is an assessed lab. You must write up and submit your answers to this lab for assessment as part of your logbook.

Learning objectives

The aims of this lab are to:

- Use shell scripting to carry out network traffic analysis, combining regular expressions and redirection
- Explore "spoofing" a MAC address, DNS lookup and the routing table

By the end of this lab session, you should be able to:

- Understand UNIX commands involving regular expressions and redirection
- Temporarily spoof the MAC address of a Linux system
- Use shell scripting to find the IP address of all machines alive on the same network

Introduction

Networking is the act of interconnecting machines to form a network so that the machines can interchange information. The most widely used networking stack is TCP/IP, where each node is assigned a unique IP address for identification. There are many parameters in networking, such as subnet mask, route, ports, host names, and so on which require a basic understanding to follow.

Several applications that make use of a network operate by opening and connecting to something called ports, which denote services such as data transfer, remote shell login, and so on. Several interesting management tasks can be performed on a network consisting of many machines. Shell scripts can be used to configure the nodes in a network, test the availability of machines, automate execution of commands at remote hosts, and so on.

Login to the Slackware machine as root.

Printing the list of network interfaces

Here is a one-line command sequence to print the list of network interfaces available on a system:

```
ifconfig | cut -c-10 | tr -d ' ' | tr -s '\n'

eth0
lo
wlan0
```

The first 10 characters of each line in the ifconfig output is reserved for writing names of network interfaces. Hence, we use:

- cut to extract the first 10 characters of each line
- tr -d ' ' to delete every space character in each line
- tr -s '\n' to squeeze the \n newline character, producing a list of interface names

Depending on how the networking is set up on your Slackware virtual machine, you will normally see either **eth0** or **wlan0**, but not both. **eth0** will be used for the rest of this lab, but if you see **wlan0**, change the commands appropriately.

Displaying IP addresses

The ifconfig command displays details of every active network interface available on the system. However, we can restrict it to a specific interface using:

```
ifconfig interface_name
```

For example:

```
ifconfig eth0

eth0
Link encap:Ethernet
HWaddr 00:1c:bf:87:25:d2        [Hardware (MAC) address]
inet addr:192.168.0.82          [IP address]
Bcast:192.168.3.255             [Broadcast address]
Mask:255.255.252.0              [Subnet mask]
```

In several scripting contexts, we may need to extract any of these addresses from the script for further manipulations. Extracting the IP address is a frequently needed task. In order to extract the IP address from the ifconfig output one could use:

```
ifconfig eth0 | egrep -o "inet addr:[^ ]*" | grep -o "[0-9.]*"

192.168.0.82
```

Here, the first command egrep -o "inet addr:[^]*" will print inet addr:192.168.0.82. The pattern starts with inet addr: and ends with some non-space character sequence (specified by [^]*). In the next pipe, it prints the character combination of digits and "**.**".

Spoofing the hardware address (MAC address)

In certain circumstances where authentication or filtering of computers on a network are based on the hardware address, we can use hardware address spoofing. The hardware address appears in the ifconfig output as HWaddr 00:1c:bf:87:25:d2.

We can spoof the hardware address at the software level as follows:

```
ifconfig eth0 hw ether 00:1c:bf:87:25:d5
```

In the preceding command, 00:1c:bf:87:25:d5 is the new MAC address to be assigned. This can be useful when we need to access the Internet through MAC-authenticated service providers that provide access to the Internet for a single machine. However, note that this only lasts until a machine restarts.

DNS lookup

There are different DNS lookup utilities available from the command line, which will request a DNS server for an IP address resolution. host and nslookup are two of such DNS lookup utilities. When host is executed it will list out all of the IP addresses attached to the domain name. nslookup is another command that is similar to host, which can be used to query details related to DNS and resolving of names. For example:

```
host google.com

google.com has address 64.252.191.242
[...]
google.com has address 64.252.191.217
```

```
google.com has IPv6 address 2a00:1450:4009:80d::200e
google.com mail is handled by 30 alt2.aspmx.l.google.com.
google.com mail is handled by 50 alt4.aspmx.l.google.com.
google.com mail is handled by 40 alt3.aspmx.l.google.com.
google.com mail is handled by 10 aspmx.l.google.com.
google.com  mail  is handled  by  20 alt1.aspmx.l.google.com.
```

We can also list out all the DNS resource records as follows:

```
nslookup google.com

Server:      192.168.118.2
Address:   192.168.118.2#53

Non-authoritative answer:
Name:  google.com
Address: 64.252.191.217
[...]
Name:  google.com
Address:    64.252.191.242
```

The last line in the preceding command-line snippet corresponds to the default name server used for resolution.

Without using the DNS server, it is possible to add a symbolic name to the IP address resolution just by adding entries into the file **/etc/hosts**. In order to add an entry, use the following syntax:

```
echo ip_address symbolic_name >> /etc/hosts
```

For example:

```
echo   192.168.0.9   backupserver   >> /etc/hosts
```

After adding this entry, whenever resolution to **backupserver** occurs, it will resolve to 192.168.0.9.

Showing routing table information

Having more than one network connected with each other is a very common scenario. An example of this is in a college, where different departments may be on separate networks. In this case, when a device on one network wants to communicate with a device on the other network, it needs to go through a device which is common to the two networks. This special device is called a **gateway** and its function is to route packets to and from different networks.

The operating system maintains a table called the **routing table**, which contains the information on how packets are to be forwarded through machines on the network. The routing table can be displayed as follows:

```
route

Kernel  IP  routing  table
Destination     Gateway         Genmask         Flags   Metric  Ref    Use  Iface
192.168.0.1     *               255.255.252.0   U       2       0        0wlan0
link-local      *               255.255.0.0     U       1000    0        0wlan0
default         p4.local        0.0.0.0         UG      0       0        0wlan0
```

Or:

```
route -n

Kernel IP routing table
Destination     Gateway         Genmask         Flags Metric Ref    Use Iface
192.168.0.1     0.0.0.0         255.255.252.0   U     2      0        0wlan0
169.254.0.0     0.0.0.0         255.255.0.0     U     1000   0        0wlan0
0.0.0.0         192.168.0.4     0.0.0.0         UG    0      0        0wlan0
```

The -n option of the route command causes all entries to display numerical IP addresses rather than symbolic hostnames.

A default gateway is set as follows:

```
route add default gw ip_address interface_name
```

For example:

```
route add default gw 192.168.0.1 wlan0
```

Listing all the machines alive on the network

When we deal with a large local area network, we may need to check the availability of other machines in the network. A machine may not be alive due to one of two conditions: either it is not powered on, or due to a problem in the network. By using shell scripting, we can easily find out and report which machines are alive on the network.

Exercise 9.1

Create a bash script called **ping.sh** with the following code, then run it. Press [Ctrl] + [z] to quit the program.

```
#!/bin/bash
# Change base address 192.168.0 according to your network.
for ip in 192.168.0.{1..255};
do
        ping $ip -c 2 &> /dev/null;
        if [ $? -eq 0 ];
        then
                echo $ip is alive
        fi
done
```

Think about how the script works before reading the next section.

We used the ping command to find out the alive machines on the network. We used a for loop for iterating through a list of IP addresses generated using the expression 192.168.0.{1..255}. The {start..end} notation will expand and will generate a list of IP addresses, such as 192.168.0.1, 192.168.0.2, 192.168.0.3 up to 192.168.0.255.

ping $ip -c 2 &> /dev/null will run a ping command to the corresponding IP address in each execution of the loop. The -c option is used to restrict the number of echo packets to be sent to a specified number. &> /dev/null is used to redirect both stderr and stdout to **/dev/null** so that it won't be printed on the terminal. Using $? we evaluate the exit status. If it is successful, the exit status is 0, else it is non-zero. Hence, the IP addresses which replied to our ping are printed.

In this script, each ping command for the IP address is executed one after the other. Even though all the IP addresses are independent of each other, the ping command is executed as a sequential program, it takes a delay of sending two echo packets and receiving them or the time-out for a reply for executing the next ping command.

10 * Further UNIX tools

This is an assessed lab. You must write up and submit your answers to this lab for assessment as part of your logbook.

Learning objectives

The aims of this lab are to:

- Gain experience in using "classic" UNIX user and system management tools
- Explore symlinks and further file management commands
- Introduce file compression and backup utilities built into UNIX

By the end of this lab session, you should be able to:

- Use tools such as finger to find out more information about a UNIX user
- Create symlinks and hard links, and explain the difference between the two
- Compress files into archives, and then extract them

Log in to the Slackware machine as root.

User and system information

users and who show the list of users logged into a machine. who also shows the terminal they are using and the date they logged in on.

```
users
```

```
bob
```

```
who
```

```
bob          tty1         2014-12-05  19:41
```

w gives more detailed information than who.

```
w
```

```
USER     TTY      FROM             LOGIN@  IDLE   JCPU   PCPU  WHAT
bob      tty1                      08:45    1.00s  0.06s  0.00s w
```

finger also gives detailed information about the list of users logged into a machine, but it can also give detailed information about a specific user (whether or not they are currently logged in).

```
finger ashley
```

```
Login name:          ashley
Registered name:     Dr A. Smith          Directory:    /home/ashley
Personal name:       Ashley Smith         Shell:        /bin/bash
Affiliation(s):      Department of Computer Science
Last login Tue Aug 6 18:24 2013 (BST) on /dev/pts/0 from localhost
Mail last read Tue Aug  6 18:36 2013 (BST)
Plan:
To think of a plan.
```

In order to get information about previous boot and user login sessions, use the last command.

```
last
```

```
smith    tty2                          Fri Feb  5 10:31   still logged in
bob      tty1                          Fri Feb  5 08:45   still logged in
reboot   system boot  3.19.0-25-generi Fri Feb  5 08:45 - 10:48  (02:02)
bob      tty1                          Thu Feb  4 20:07 - down   (01:48)
reboot   system boot  3.19.0-25-generi Thu Feb  4 20:07 - 21:55  (01:48)

wtmp begins Thu Feb  4 20:07:31 2016
```

last can also be used to obtain information about login sessions for a single user or just the reboot sessions.

```
last bob
```

```
bob      tty1                          Fri Feb  5 08:45   still logged in
bob      tty1                          Thu Feb  4 20:07 - down   (01:48)

wtmp begins Thu Feb  4 20:07:31 2016
```

```
last  reboot
```

```
reboot   system boot  3.19.0-25-generi Fri Feb  5 08:45 - 10:48  (02:02)
reboot   system boot  3.19.0-25-generi Thu Feb  4 20:07 - 21:55  (01:48)

wtmp begins Thu Feb  4 20:07:31 2016
```

Information about failed user logins (if any) can be displayed by executing lastb.

```
lastb
```

```
bob      tty1                          Fri Feb  5 10:54 - 10:54  (00:00)
UNKNOWN  tty1                          Fri Feb  5 10:53 - 10:53  (00:00)
root     tty1                          Fri Feb  5 10:52 - 10:52  (00:00)

btmp begins Fri Feb  5 10:57:16 2016
```

In order to see how long the system has been powered on, use the uptime command.

```
uptime
```

```
12:33  up 12 days, 20:57, 2 users, load averages: 1.79, 1.84, 1.78
```

Exercise 10.1 User and system information

Before attempting the questions below, you may wish to deliberately reboot the machine and create some failed login attempts so that you have some data to work with.

1. How many login attempts (successful and failed) occurred in the past 48 hours?
2. How many system reboots occurred in the past 48 hours?

Symbolic and hard links

In the * Email under Linux lab, you were introduced to the mail command. In Slackware 13.37, the program is actually called mailx. Some (older) Linux distributions use nail instead.

So how can we tell whether to use mail, mailx or nail? The command

```
whereis program
```

tells you the path to the binaries in the system, so the command

```
whereis mailx
```

will tell you that mailx is in **/usr/bin/mailx** (and also return a number of other answers). If you do the same for mail, you should find that it returns **/bin/mail**. If you execute

```
ls -l /bin/mail
```

you will see that **/bin/mail** is a symlink (symbolic link) to **/usr/bin/mailx**. Therefore it does not matter if we execute mail or mailx: they are exactly the same. You may wish to try the above for nail as well.

Symlinks are similar to shortcuts in Microsoft Windows and aliases in Mac OS X. Symlinks may be created using the syntax below:

```
ln -s /path/to/original/file /path/to/symlink
```

Hard links may be created as above, but without the -s option.

Exercise 10.2 Symbolic and hard links

1. Create a file **~/unixstuff/extra_file** and a symlink **~/unixstuff/links/extra_file_link** which links to **extra_file** (you may need to create the **links** directory). Use `ls -l` whilst in **~/unixstuff/links/** to check that the symlink has been created.
2. Edit **extra_file** and add some text to it. Now open **extra_file_link** by executing the following command:

```
cat ~/unixstuff/links/extra_file_link
```

   Do you see the changes you made?

3. Move **extra_file** to the **backups** directory (so its location is now **~/unixstuff/backups/extra_file**).
   a) What happens to **extra_file_link** (if anything)? **Hint:** try opening the symlink using cat, what is the result? Execute `ls -l` whilst in **~/unixstuff/links/**, do you notice anything different?
   b) Move **extra_file** back to the **unixstuff** directory – predict what happens to **extra_file_link** then test your prediction.
4. Delete **extra_file_link**. What happens to **extra_file** (if anything – try opening it using cat)?
5. Recreate the **extra_file_link** symlink and delete **extra_file**. What happens to **extra_file_link** (if anything)? See the hint to question 3 (a) if you are stuck.
6. Delete **extra_file_link** and redo questions 1 – 5 above, but this time use hard links instead. Hence explain the differences between symbolic and hard links. You might also wish to do some research to explain why you see these differences.

File management

Some useful file management commands are:

- df (disk free)

  The df command outputs a report on the disk space available.

- diff

  diff allows you to compare the content of two text files and outputs the differences. The syntax is as follows:

  diff *file1 file2*

  Lines in **file1** are prefixed with < whilst lines in **file2** are prefixed with >.

- find

  find searches through the file system for files matching specified attributes (such as file name, file contents, size). Execute man find to see what options are available.

- touch

  touch updates the access and modification date and time of a file to the current time. The syntax is as follows:

  touch *file*

  If **file** does not already exist, it is created with zero contents.

File compression and backup

UNIX systems usually support a number of utilities for backing up and compressing files. The most useful are:

- tar (tape archiver)

  tar backs up entire directories and files onto a tape device or (more commonly) into a single disk file known as an archive. An archive is a file that contains other files plus information about them, such as their filename, owner, timestamps, and access permissions. tar does not perform any compression by default.

  To create a disk file tar archive, use

  tar -cvf *archivename filename*

  where *archivename* will usually have a **.tar** extension. Here the -c option means create, -v means verbose (output filenames as they are archived), and -f means file. To list the contents of a tar archive, use

  tar -tvf *archivename*

  To restore files from a tar archive, use

  tar -xvf *archivename*

- cpio

  cpio is another facility for creating and reading archives. Unlike tar, cpio doesn't automatically archive the contents of directories, so it's common to combine cpio with find when creating an archive:

  find . -print -depth | cpio -ov -H tar > *archivename*

  This will take all the files in the current directory and the directories below and place them in an archive called *archivename* . The -depth option controls the order in which the filenames are produced and is recommended to prevent problems with directory permissions when doing a restore. The -o option creates the archive, the -v option prints the names of the files archived as they are added and the -H option specifies

an archive format type (in this case it creates a tar archive). Another common archive type is crc, a portable format with a checksum for error control.

To list the contents of a cpio archive, use

cpio -tv < *archivename*

To restore files, use:

cpio -idv < *archivename*

Here the -d option will create directories as necessary. To force cpio to extract files on top of files of the same name that already exist (and have the same or later modification time), use the -u option.

- compress and gzip

compress and gzip are utilities for compressing and decompressing individual files (which may be or may not be archive files). To compress files, use:

compress *filename* or gzip *filename*

In each case, *filename* will be deleted and replaced by a compressed file called **filename.Z** or **filename.gz**. To reverse the compression process, use:

compress -d *filename* or gzip -d *filename*

zcat can be used to read gzipped text files without uncompressing them first. The output can be piped to less if the text file is very long.

Exercise 10.3 File compression and backup

1. Archive the contents of your home directory (including any subdirectories) using tar and cpio.
2. Compress the tar archive with compress, and the cpio archive with gzip.
3. Now extract their contents.
