LFS216

Linux Security Fundamentals

Version 1.0

LFS216: Version 1.0

Published by:

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

Contents

Chapter 1

Introduction

1.1 Labs

Lab 1

Exercise 1.1 Lab There is no lab to complete for this chapter.

Chapter 2

Security Basics

2.1 Labs

Lab 2

Exercise 2.1 Installing and Running Virtual Machines In this exercise we will be installing and configuring the virtual machines that we use for subsequent lab exercises.

We will be using two virtual machines for most lab exercises, they are going to be called `main.example.com` and `secondary.example.com`. We will also be using an **appliance VM** which will automatically configure on **VMWare** and **Oracle Virtual Box** hypervisors. The appliance virtual machine uses very little memory and disk space.

There is an optional exercise that installs a pre-configured Operating System and application on to the **temporary VM**, however, this installed appliance requires a minimum of 8GB of memory for the **VM**. The optional exercise with the large memory requirement is clearly marked.

The virtual machines will require **two network** connections. One connection will be the default network created automatically when the virtual machine is created. The second network is completely virtual. Most hypervisors **VMWare,Oracle Virtual Box, Parallels** have the capability to create **private** or **host only** networks. These private networks can be configured so the virtual machines are all on a single private network. Be advised that you may have to manually add an additional virtual network adapter to the virtual machines.

DHCP addresses are assigned by the virtualization system, it will be necessary to update `/etc/hosts` on both `main.example.com` and `secondary.example.com`. Be sure to set the system's current IP address in the corresponding entries in `/etc/hosts` on **both** hosts.

Since the virtual appliance will create its own virtual machine We only need the two machines `main.example.com` and `secondary.example.com` created.

Resources: Here is where the resources can be found for this class.

- The userid is **LFtraining** and password is **Penguin2014** for the items on the `https://training.linuxfoundation.org/` website.
- The Virtual Machine Images can be found at:
  `https://training.linuxfoundation.org/cm/VIRTUAL_MACHINE_IMAGES/`
  Please use the CentOS7 image for this class.
- Tools used during class are found at:
  `https://training.linuxfoundation.org/cm/LFS216`
  There is a "solutions" tar file containing extra information and sample files for the exercises. This is the same file that the **ready-for.sh** will prompt to download.

Also in this location are **iso images** and larger files required for class. Most of these are available on the Internet but are place here for convenience and are the versions tested for class.

- Tool for verify your VMs are ready for class.
  `https://training.linuxfoundation.org/cm/prep/`
  This is the location for the **ready-for.sh** script. Run this script on the two Vim's after they are running.

The lab exercises have been created and tested on the Linux Foundation's **CentOS7** virtual machines. Other Linux distributions have not been tested at this time.

Steps to create the systems `main.example.com` and `secondary.example.com` on your existing Hypervisor:

- Download the **CentOS7** image from:
  `https://training.linuxfoundation.org/cm/VIRTUAL_MACHINE_IMAGES/`
- Expand the `CentOS7.tar.gz` image.
- Create the virtual machine in the hypervisor using a pre-existing disk. (the one from the `CentOS7.tar.gz` image.
- Check the **000README** file located at `https://training.linuxfoundation.org/cm/VIRTUAL_MACHINE_IMAGES/000README`
- Start the **VM** and verify/correct the network if necessary.
- Shutdown the **VM**
- Use the **clone** function in the hypervisor, if an option, initialize or change the network **MAC** addresses.
- You should have two identical systems, set the hostnames to be `main.example.com` and `secondary.example.com`.
- Run the command `/home/student/LFT/ready-for.sh\LFS216` on each machine.

Solution 2.1

See lab exercise instructions. Make sure you are using an ISOLATED virtual network for all virtual machines in this course.

Network Information:

`https://www.vmware.com/support/ws55/doc/ws_net_configurations_hostonly.html`

`https://www.virtualbox.org/manual/ch06.html`

`http://kb.parallels.com/4948`

Clone Information:

`https://www.virtualbox.org/manual/ch01.html#clone`

`https://www.vmware.com/support/ws55/doc/ws_clone_overview.html`

`http://download.parallels.com/desktop/v9/ga/docs/pt_BR/ParallelsDesktopUser'sGuide/32894.htm`

Chapter 3

Threats and Risk Assessment

3.1 Labs

Lab 3

Exercise 3.1 Introduction to **tcpdump** and **wireshark**

In this exercise, we learn about two of the most useful tools for troubleshooting what is happening as network traffic is transmitted and received on the system: **tcpdump** and **wireshark**. These are *passive* tools; they simply listen to all traffic exposed to the system by the networking infrastructure.

A fair amount of network traffic is broadcasted to all the devices that are connected to the networking gear. Much of that is simply dumped by the system because the traffic's destination does not match the system's address. This traffic can be analyzed With **tcpdump** and **wireshark**.

tcpdump is a command-line, low level tool that is generally available as part of a **Linux** distribution's default package installation. **tcpdump** has a filtering capability as described in the **pcap-filter** man page; both **tcpdump** and **wireshark** use the **pcap** libraries to capture and decipher traffic data.

tcpdump lacks a graphical component as well as the ability to analyze the traffic it captures. For this reason, it is typically used to capture network traffic during an interesting session and then the resulting capture files are copied to a workstation for analysis using the **wireshark** utility. Packet capture also requires placing the network interfaces into promiscuous mode, which requires root permissions.

On `main.example.com`, open a terminal and run the command:

```
# tcpdump -D
```

then to see which adapter is our inside adapter run:

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

This will show the available interfaces on which you can capture network traffic. Let's use `enp0s8`.

```
# tcpdump -i enp0s8
```

This will print a brief summary of each packet that the system sees on the interface, regardless of whether it is intended for `main.example.com`. Leave the process running and open a second terminal on `secondary.example.com`. In this second terminal, run **ping**, first pinging `main.example.com` and then pinging the broadcast address,(this is the same network as your adapter but with a host number of "255", something like `192.168.56.255`.

```
$ ping -c4 main.example.com
$ ping -c4 -b  192.168.56.255
```

Next, explore the **pcap-filter** and **tcpdump** man pages. We are going to construct a **tcpdump** command that captures HTTP traffic on `enp0s8` that is coming from `secondary.example.com` and save that traffic to a file. On `main.example.com`, run the following commands:

```
# yum install httpd
# systemctl start httpd
# echo "test page" > /var/www/html/index.html
# firewall-cmd  --zone=public --add-port=80/tcp --permanent
# firewall-cmd --reload
# tcpdump -w /tmp/secondary-http.pcap -i enp0s8 port 80 and host secondary.example.com
```

On `secondary.example.com`, let's generate some HTTP traffic:

```
# elinks -dump http://main.example.com/index.html
```

Next, we will analyze the capture with **wireshark**. If

```
# which wireshark
```

fails, you will have to install the utility. On **RHEL**-based systems:

```
# yum install wireshark wireshark-gnome
```

You can launch it by running `/usr/sbin/wireshark` or finding it the application menus on your desktop, e.g., under `Applications -> Internet` menu, you may find the `Wireshark Network Analyzer`.

Once the application is open, go to the `File -> Open` dialog and browse to the capture file created above. Explore the various frames in the application interface.

Exercise 3.2 Introduction to **nmap**

nmap is another essential tool for troubleshooting and discovering information about the network and services available in an environment. This is an *active* tool (in contrast to **tcpdump** and **wireshark**) which sends packets to remote systems in order to determine information about the applications running and services offered by those remote systems.

Be sure to inform the network security team as well as obtain written permission from the owners and admins of the systems which you will be scanning with the **nmap** tool. In many environments, active scanning is considered an intrusion attempt.

The information gleaned from running **nmap** can provide clues as to whether or not a firewall is active in between your system and the target. **nmap** also indicates what the target operating system might be, based on fingerprints of the replies received from the target systems. Banners from remote services that are running may also be displayed by the **nmap** utility.

First, let's install **nmap** on both `main.example.com` and `secondary.example.com`. Run the following command as root on both systems:

```
# yum install nmap
```

Next, explore the **nmap** man page.

```
$ man nmap
```

Now, we will run **nmap** on each system against the other:

```
# nmap main.example.com
# nmap secondary.example.com
```

By adding the `-A` option to the **nmap** program, we can see the OS fingerprint detection capabilities of **nmap**:

```
# nmap -A secondary.example.com
```

A common usage for **nmap** is to perform a network **ping scan**; basically, **ping** all possible IP addresses in a subnet range in order to discover what IP addresses are currently in use. This is also sometimes referred to as **network discovery**.

```
# nmap -sP 192.168.56.0/24
```


Chapter 4

Physical Access

4.1 Labs

Lab 4

Exercise 4.1 Single User Mode

In this exercise, you will bypass system authentication and boot the system into single user mode in order to reset the root password. This mode is useful when either system authentication or filesystem configuration is not correct.

By interrupting the boot sequence, it is possible to add a kernel boot parameter that is then passed to the init process to alter the boot behavior. In **SysV** and **Upstart** systems a parameter of "1" or "s" presents a root level shell. The **systemd** systems will need to replace the init process by adding "init=/bin/bash" to the kernel line. If the system is using **dracut** to create the **initial ramdisk** there are command line options available at boot time (see **man 7 dracut.cmdline**). One option is the **rd.break** option which will stop the boot sequence right before control is passed to the **init** program. In this instance the root filesystem is mounted read-only on /sysroot. The root filesystem can be re-mounted read-write and chroot to the root filesystem to make changes.

Grub, the bootloader, provides on-screen instructions on editing the boot stanza. **READ THEM CAREFULLY**.

1. Reboot the virtual machine.
2. Interrupt the **grub** boot sequence by hitting any key.
3. Follow the on-screen instructions to edit the *default* boot stanza to boot into single user mode.
4. Follow the instructions on-screen to accept the changes.
5. Follow the on-screen instructions to boot the system.
6. Once the system is in single user mode, set the filesystem to read/write by issuing the command:

   ```
   # mount -o remount,rw /
   ```

7. Set the root password to `linuxfoundation`
8. Exit the single user mode.
9. Repeat the exercise and set the root password to `LFtrain`
10. If the systems is using **SELinux**, touch the file `/.autorelabel` and reboot to correct any changed to the file contexts.

Exercise 4.2 Disk Encryption

In this exercise, you will encrypt a partition on the disk in order to provide a measure of security in the event that the hard drive or laptop is stolen.

Linux provides **cryptsetup**, an encryption utility that takes advantage of **Device Mapper** to provide data encryption at the block layer. **cryptsetup** creates a virtual block device that is the un-encrypted pass-through device where the file system is created and accessed.

1. Review the **man** page for **cryptsetup**.
2. Create a new 100M partition for the encrypted block device.
3. Changes to the partition table will require rebooting.
4. Format the partition with **cryptsetup**, using **LUKS** for the crypto layer.
5. Create the un-encrypted pass through device by opening the crypted block device, i.e., `secret-disk`.
6. Add an entry to `/etc/crypttab` so that the system prompts for the passphrase on reboot.
7. Format the filesystem as an **ext4** filesystem.
8. Create a mount point for the new filesystem, ie. `/secret`.
9. Add an entry to `/etc/fstab` so that the filesystem is mounted on boot.
10. Test the `/etc/fstab` entry for syntax, review the **man** page for **mount** to see what syntax will mount all filesystems.
11. Validate the entire configuration by rebooting.

Solution 4.2

2. `# fdisk -cu /dev/sda`
3. `# reboot`
4. `# cryptsetup luksFormat /dev/sda4`
5. `# cryptsetup luksOpen /dev/sda4 secret-disk`
6. Add the following to `/etc/crypttab`:

   ```
   secret-disk         /dev/sda4
   ```

7. `# mkfs -t ext4 /dev/mapper/secret-disk`
8. `# mkdir -p /secret`
9. Add the following to `/etc/fstab`:

   ```
   /dev/mapper/secret-disk    /secret    ext4    defaults    1 2
   ```

10. `# mount -a`
11. `# reboot`

Exercise 4.3 Swap / Encrypted Swap

In this exercise, we will be encrypting the swap partition. Data written to the swap device can contain sensitive information. Since swap is backed by an actual partition, it is important to consider the security implications of having this partition unencrypted.

The process for encrypting this partition is similar to the previous exercise with the modification that we are not creating a file system on the encrypted block device.

In this case, we are also going to use the existing the swap device by first de-activating it and then formatting it for use as an encrypted swap device.

1. Review the **man** page for **cryptsetup**.
2. Review the **man** page for **swapoff** and de-activate the inactive swap device (ours is `/dev/sda3`, yours may be different).

3. Add an entry to /etc/crypttab so that the system prompts for the passphrase on reboot.
4. Modify the **swap** entry in /etc/fstab so that the swap devices is activated on boot.
5. Check the fstab entry for syntax, review the **man** page for **swapon** to see what syntax will activate all swap devices.
6. Validate the entire configuration by rebooting.

Solution 4.3

2.
```
# swapon -s
# swapoff /dev/sda3
```

3. Add the following to /etc/crypttab:

```
swap     /dev/sda3    /dev/urandom  swap,cipher=aes-cbc-essiv:sha256,size=256
```

4. Modify the existing **swap** entry in /etc/fstab:

```
/dev/mapper/swap    swap    swap    defaults    0 0
```

5. Verify there is only one **swap** entry in the /etc/fstab
6.
```
# reboot
...
$ swapon -s
```


Chapter 5

Logging

5.1 Labs

Lab 5

Exercise 5.1 **rsyslog** Remote Logging

In this lab, we will configure a remote logging service and a client that logs both locally and remotely. Logging to both location is important for the ability to audit and verify logs are accurate. This configuration also provides a limited ability to recreate an attack based on when logs diverged if a client becomes compromised.

It may be necessary to open some ports on the firewall on `main.example.com` in order to receive messages from `secondary.example.com`. Run the following command on `main.example.com`:

```
# firewall-cmd --zone=public --add-port=514/tcp --permanent
# firewall-cmd --zone=public --add-port=514/udp --permanent
# firewall-cmd --reload
```

1. Review the `rsyslog.conf` **man** page.
2. Edit the `/etc/rsyslog.conf` on the `main.example.com` system such that it now receives messages from network clients.
3. Be sure to restart the **rsyslog** daemon when you modify its configuration.
4. Use the **logger** utility to inject test messages into the logging system.
5. Edit the `/etc/rsyslog.conf` on the `secondary.example.com` system to send messages to the service.
6. Again, be sure to restart the **rsyslog** daemon when changing the configuration.
7. Test the configuration by using **logger** and verify that messages are found locally and on the server.

Solution 5.1

Make sure you have first opened the firewall ports on `main.example.com`:

```
# firewall-cmd --zone=public --add-port=514/tcp --permanent
# firewall-cmd --zone=public --add-port=514/udp --permanent
# firewall-cmd --reload
```

The rsyslog.conf **man** page and the default rsyslog.conf file itself contain the necessary configs, commented out, that we need in order to accomplish this task. Uncommenting the appropriate lines will achieve the desired configuration.

2. On `main.example.com`, edit the `/etc/rsyslog.conf` such that the following lines are uncommented:

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

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

3. Then restart **rsyslog**:

```
# service rsyslog restart
```

4. and generate a test message:

```
$ logger -p local0.info 'test logging'
```

5. On `secondary.example.com`, edit `/etc/rsyslog.conf` such that the following lines have been added in the `#### RULES ####` section:

```
# Provides UDP forwarding
*.*   @192.168.1.10

# Provides TCP forwarding
*.*   @@192.168.1.10
```

6. `$ service rsyslog restart`

7. Test by using the **logger** utility on the client, `secondary.example.com`, and view the message on the server, `main.example.com`.

8. Please undo the changes and restart **rsyslog**

Exercise 5.2 Syslog Protocol Analysis with **wireshark**

In this lab, we will use **wireshark** to examine the **syslog** protocol across the network. Here we will see that the standard **syslog** protocol is un-encrypted, un-authenticated, and un-verified. By using TCP we gain some measure of verification, by adding SSL encryption, we add a measure of data integrity and privacy.

1. Make sure the previous exercise is working; i.e., messages from `secondary.example.com` are being logged on `main.example.com`.
2. Launch **wireshark** and view the traffic in real-time from the server. Observe the difference in the communication streams by turning off TCP, monitoring, re-enabling TCP and disabling UDP message forwarding on the client system.
3. To enable TLS encryption for the transport stream. It will be necessary to install the appropriate packages to enable the **rsyslog-gnutls** extensions. On EL and **Ubuntu** systems, this is **rsyslog-gnutls**.
4. A certificate and its corresponding key have been created and are available in the **solutions** directory. There are three files provided that contain a set of certificates, corresponding **server** configuration file for **main.example.com** and a client configuration file for **secondary.example.com**. To use these configuration files, comment out the changes that have been made to the `/etc/rsyslog.conf` file on **main.example.com** and **secondary.example.com**, returning them to their original state. Then copy the related file form the solutions directory to the `/etc/rsyslog.d` directory. Examine the new configuration file on **main.example.com** and copy the provided key and certificate files to the locations indicated. `main.example.com`, in the `/etc/pki/tls/certs/` directory. Use `rsyslog.crt` as both the certificate and CA file. `rsyslog.key` is the key file. It will also be necessary to disable the firewall on `main.example.com` in order to receive messages from `secondary.example.com`. Run the following command on `main.example.com`:
5. Restart the rsyslog service to enable the changes.
6. Ensure the **TCP port 6514** is not blocked by the firewall on **main.example.com**.

Solution 5.2

1. Verify **logger** messages are being sent from **secondary.example.com** to **main.example.com**.

```
[secondary]$ logger "Yes the remote logging is working"
[main]# tail /var/log/messages
```

2. The system **main.example.com** should be listening for **TCP** and **UDP** remote logging messages. Changing the target in the file `/etc/rsyslog/rsyslog.conf` for sending remote log messages on **secondary.example.com** will switch the protocol as desired.

```
*.*   @<ip of main>  # send via UDP
*.*  @@<ip of main>  # send via TCP
```

3. Verify or install **rsyslog-gnutls** is installed.

```
# yum -y install rsyslog-gnutls
```

4. Comment out any changed made to the `/etc/rsyslog.conf` file. Copy the files from the **solutions** directory to the correct locations.

```
[main]# cp <solutions-dir>/main-tls.conf /etc/rsyslog.d/main-tls.conf
[main]# cd /etc/pki/tls/certs
[main]# tar xvf <solutions-dir>/lab-keys.tar
[secondary]# cp <solutions-dir>/secondary-tls.conf /etc/rsyslog.d/secondary-tls.conf
[secondary]# cd /etc/pki/tls/certs
[secondary]# tar xvf <solutions-dir>/lab-keys.tar
```

Another option is to edit the files manually. The parameters that are suggested and used in the solutions files are below:

For the remote log server (main.example.com), the following edits are installed with the file `main-tls.conf`:

```
## Added for TLS support
# make gtls driver the default
$DefaultNetstreamDriver gtls

# certificate files
$DefaultNetstreamDriverCAFile   /etc/pki/tls/certs/lab.crt
$DefaultNetstreamDriverCertFile /etc/pki/tls/certs/lab.crt
$DefaultNetstreamDriverKeyFile  /etc/pki/tls/certs/lab.key

$ModLoad imtcp # load TCP listener

$InputTCPServerStreamDriverMode 1 # run driver in TLS-only mode
$InputTCPServerStreamDriverAuthMode anon # client is NOT authenticated
$InputTCPServerRun 6514 # start up listener at port 6514

## End TLS additions
```

The matching remote logging client uses the options as in the file `secondary-tls.conf`:

```
# This is the client side of the TLS encrypted rsyslog

# certificate file, just the CA file for a client
$DefaultNetstreamDriverCAFile /etc/pki/tls/certs/lab.crt

# set up action
$DefaultNetstreamDriver gtls           #use the gnutls netstream driver
$ActionSendStreamDriverMode 1          #require the use of tls
$ActionSendStreamDriverAuthMode anon   #the server is NOT authenticated
# send all messages
*.* @@(o)main.example.com:6514
```

These files were created following the guide found at `http://www.rsyslog.com/doc/rsyslog_tls.html`. Self-signed SSL keys were used in this lab for demonstration purposes.

5. Restart the **rsyslog** service on **main** then **secondary**.

```
# systemctl restart rsyslogd
```

6. Ensure the firewall allows encrypted log traffic.

```
# firewall-cmd --zone=public --add-port=6514/udp --permanent
# firewall-cmd --reload
```

7. Now run **wireshark** again and observe the traffic on port 6514.

Note: For additional information on the generation of SSL certificates:

- For generating the SSL certificate, use the following:

```
# cd /etc/pki/tls/certs
# make rsyslog.crt
# mv rsyslog.key rsyslog-pass.key
# openssl rsa -in rsyslog-pass.key -out rsyslog.
```

- Explanation: In the `/etc/pki/tls/certs` directory, there is a `Makefile` that uses the **openssl** commands to generate a variety of SSL certificate files. Run **make** while in the `/etc/pki/tls/certs` directory for additional help.
- Once the certificate is created, it will be useful to remove the passphrase from the key file so that an operator is not required to unlock the keyfile if the service is restarted. The resulting `crt` and `key` files are then the public and private pairs of the SSL/TLS x509 certificate.
- This process will be used later in this course to generate certificates handy for other uses.

Exercise 5.3 Logwatch Report

This exercise will explore some of the options in **logwatch**. In its default configuration **logwatch** will send an email to the **root** user on a daily basis. The **MailTo** option can be modified to send the report to another user. It is most common the **logwatch** be run as a daily background process but it also has an interactive option. Consult the **man logwatch** information as required.

Verify **logwatch** is installed.

Using an interactive **logwatch** command:

1. Use an interactive **logwatch** command verify if your system has some log records.
2. Use an interactive **logwatch** command verify if your system has some log records from yesterday.
3. Use an interactive **logwatch** command display any log records for **sudo** service.
4. Use an interactive **logwatch** command to display any log records for the **sudo** service with **high** detail level.
5. Customize **logwatch** so it does not report log entries from the **smartd** service
6. Customize **logwatch** to send a daily report to user `student` as well as user `root`.
7. Customize **logwatch** to increase the reporting detail for the **sudo** to **high**.

Solution 5.3 Verify **logwatch** is installed.

```
# yum install logwatch
```

Using an interactive **logwatch** command:

1. Use an interactive **logwatch** command verify if your system has some log records. (be patient)

```
# logwatch --output stdout --range all
```


2. Use an interactive **logwatch** command verify if your system has some log records from yesterday.

```
# logwatch --output stdout --range yesterday
```

3. Use an interactive **logwatch** command display any log records for **sudo** service.

```
# logwatch --service sudo --output stdout --range all
```

4. Use an interactive **logwatch** command to display any log records for the **sudo** service with **high** detail level.

```
# logwatch --service sudo --detail high --output stdout --range all
```

5. Customize **logwatch** so it does not report log entries from the **smartd** service.

```
# echo 'Service = "-smartd"' >> /etc/logwatch/conf/logwatch.conf
```

6. Customize **logwatch** to send a daily report to users `student` as well as user `root`.

```
# echo 'MailTo = "student root"' >> /etc/logwatch/conf/logwatch.conf
```

7. Customize **logwatch** to increase the reporting detail for the **sudo** to **high**.

```
# echo "services/sudo: Detail = High" >>/etc/logwatch/conf/override.conf
```

Chapter 6

Auditing and Detection

6.1 Labs

Lab 6

Exercise 6.1 **AIDE** Monitoring of Critical Files

The Advanced Intrusion Detection Environment (**AIDE**) tool was mentioned in the Auditing and Detection unit since it falls under both categories - IDE and Local Security. The way the tool works is by creating a database of the existing files and then regularly comparing the current file attributes to those logged in the database. Since the initial database is essentially the baseline of what the system files should be, create this database shortly after installing the operating system. Be sure to update this database anytime an authorized modification has been made to the monitored files.

The basic process for setting up AIDE is as follows:

- Edit `/etc/aide.conf`
- Initialize the AIDE database, this will take a few minutes with the default configuration
- Store the db, binary, and configuration files somewhere safe
- Copy the initial database to the working db location
- Create a cron job to check the system
- Periodically check the current configuration, db, and binaries against the stored ones

To validate your configuration, edit some of the monitored files and use the `check` option.

Solution 6.1

The basic process for setting up AIDE is as follows:

```
# vim /etc/aide.conf
/usr/sbin/aide --init
# tar -zcvf initial-aide.tgz /etc/aide.conf /usr/sbin/aide /var/lib/aide/aide.db.new.gz \
  && rsync -avP initial-aide.tgz SAFEHOST:
# cp /var/lib/aide/aide.db.new.gz /var/lib/aide/aide.db.gz
# echo "/usr/sbin/aide --check" >> /etc/cron.d/aide-check && chmod 700 /etc/cron.d/aide-check
```

To validate your configuration, edit some of the monitored files and use the `check` option.

```
# echo "99.99.99.99 badhostname.delete.me" >> /etc/hosts
# touch /etc/passwd
# /usr/sbin/aide --check
```

Be sure to clean up /etc/hosts once done.

Exercise 6.2 **OSSEC** Installation and Configuration

OSSEC is an advanced host based intrusion detection platform. There are three components to the framework:

1. system agent
2. server with UI
3. rule sets

On **RPM**-based systems, Installing OSSEC can be as simple as adding the **Atomicorp** repo, and using **yum** to install the appropriate packages. Even easier, one can use the install.sh script provided by **Atmoicorp** to configure the system for the repos, and then install the appropriate packages. Detailed information for **Centos**, **Fedora** and **openSUSE** systems can be found at http://www5.atomicorp.com/channels/ossec. Install **OSSEC** server on main.example.com.

1. Acquire the **rpm** repository from **Atomicorp.com**.
2. Install the server component and the web user interface.
3. Correct email to send to **root@localhost.**, from **ossecm@localhost.** and SMTP server of **localhost.**
4. To satisfy the alert generation tool's requirement for a communication socket file, create a dummy client agent. Use the tool /var/ossec/bin/manage_agents and create an agent.
5. Restart ossec.
6. Check the /var/ossec/logs/ossec.log for any errors during start up.
7. Starting or restarting the ossec server should generate an alert email to root. Verify an email was received by root or check the /var/ossec/logs/alerts/alert.log for an "Ossec server started" alert.
8. Connect the client (agent) with the server (manager). Install **OSSEC** client on secondary.example.com:
9. Configure the client to communicate with the server. In the <server-ip> statement of the <client> stanza of the <ossec_config> section in /var/ossec/etc/ossec.conf file, set the ip address of the server (manager).
10. Create a client (agent) definition on the server with the manage_agents command.
11. Restart the ossec-hids service on the client (agent).
12. On **main.example.com** prepare a key and certificate to be used for encrypted communication.
13. On **main.example.com** temporally start the key exchange daemon **ossec-authd**.
14. Setup the shared key on the agent (secondary.example.com).
    Note: use the ip address of main.example.com.
15. For further customization on main.example.com, shorten the interval that the **syscheck** runs. The default is for **syscheck** to run every 22 hours (79200 seconds). It may be helpful to reduce that time to 60 seconds during testing. Near the top of the file /var/ossec/etc/ossec.conf is a section <syscheck> with a <frequency> stanza that defaults to 79200 seconds, change that to 60 seconds and restart **ossec-hids**
16. **Ossec** does not report newly created files in directories. This feature can be added.
    - Turn on the "new files" feature.
    - Now indicate to **syscheck** which directories to report the changes in real time by adding the report_changes="yes" and the realtime="yes" options to the directory entry in the /var/ossec/etc/ossec.conf file.
    - The addition of a new file is not normally an event of high enough priority to warrant sending an email but that can be overridden with a local rule.

    Save the file and restart ossec-hids. Add a file to the monitored directory. It may take a few minutes before the email alert is sent.

Solution 6.2

1. Acquire the **rpm** repository from **Atomicorp.com** for **main** and **secondary**.

```
# wget -q -O atomic-file  https://updates.atomicorp.com/installers/atomic
# sh atomic-file
```


2. Install the server component and the web user interface on **main**.

```
# yum -y install ossec-hids ossec-hids-server ossec-wui
```

3. Correct email to send to **root@localhost.**, from **ossecm@localhost.** and SMTP server of **localhost.**

```
# vim /var/ossec/etc/ossec.conf
```

In the section <ossec_config> locate and modify the following:

```
<email_to>daniel.cid@xxx.com</email_to>
<smtp_server>smtp.xxx.com.</smtp_server>
<email_from>ossecm@ossec.xxx.com.</email_from>
```

to:

```
<email_to>root@localhost</email_to>
<smtp_server>localhost</smtp_server>
<email_from>ossecm@localhost</email_from>
```

4. To satisfy the alert generation tool's requirement for a communication socket file, create a dummy client agent. Use the tool /var/ossec/bin/manage_agents and create an agent like below:

```
# /var/ossec/bin/manage_agents


****************************************
* OSSEC HIDS v2.8.3 Agent manager.     *
* The following options are available: *
****************************************
   (A)dd an agent (A).
   (E)xtract key for an agent (E).
   (L)ist already added agents (L).
   (R)emove an agent (R).
   (Q)uit.
Choose your action: A,E,L,R or Q: a

- Adding a new agent (use '\q' to return to the main menu).
  Please provide the following:
   * A name for the new agent: server-agent
   * The IP Address of the new agent: 127.0.0.1
   * An ID for the new agent[001]:
Agent information:
   ID:001
   Name:server-agent
   IP Address:127.0.0.1

Confirm adding it?(y/n): y
Agent added.
```

5. Restart ossec.

```
# systemctl restart ossec-hids.service
```

6. Check the /var/ossec/logs/ossec.log for any errors during start up.

```
# less /var/ossec/logs/ossec.log
```

7. Starting or restarting the ossec server should generate an alert email to root. Verify an email was received by root or check the /var/ossec/logs/alerts/alert.log for an "Ossec server started" alert.

```
# less /var/ossec/logs/alerts/alert.log
```

8. Connect the client (agent) with the server (manager). Install **OSSEC** client on secondary.example.com:

```
# yum -y install ossec-hids ossec-hids-client
```

9. Configure the client to communicate with the server. In the <server-ip> statement of the <client> stanza of the <ossec_config> section in /var/ossec/etc/ossec.conf file, set the ip address of the server (manager).

```
# vi /var/ossec/etc/ossec.conf
```

Should look like this: (with your server IP address)


```
<ossec_config>
  <client>
    <server-ip>192.168.0.29</server-ip>
  </client>
```

10. Create a client (agent) definition on the server with the `manage_agents` command.

```
# /var/ossec/bin/manage_agents

****************************************
* OSSEC HIDS v2.8.3 Agent manager.     *
* The following options are available: *
****************************************
   (A)dd an agent (A).
   (E)xtract key for an agent (E).
   (L)ist already added agents (L).
   (R)emove an agent (R).
   (Q)uit.
Choose your action: A,E,L,R or Q: a

- Adding a new agent (use '\q' to return to the main menu).
  Please provide the following:
   * A name for the new agent: remote
   * The IP Address of the new agent: 192.168.0.18
   * An ID for the new agent[1025]:
Agent information:
   ID:1025
   Name:remote
   IP Address:192.168.0.18

Confirm adding it?(y/n): y
Agent added.
```

11. Restart the ossec-hids service on the client (agent).

```
# systemctl restart ossec-hids.service
```

12. On **main.example.com** prepare a key and certificate to be used for encrypted communication.

```
# openssl genrsa -out /var/ossec/etc/sslmanager.key 2048
# openssl req -new -x509 -key /var/ossec/etc/sslmanager.key \
        -out /var/ossec/etc/sslmanager.cert -days 365
```

13. On **main.example.com** temporally start the key exchange daemon **ossec-authd**.

```
# /var/ossec/bin/ossec-authd -p 1515 >/dev/null 2>&1 &
```

14. Setup the shared key on the agent (secondary.example.com).

    Note: use the ip address of main.example.com.

```
 # /var/ossec/bin/agent-auth -m 192.168.0.23 -p 1515
```

    The response should be something like:

```
INFO: Connected to 192.168.0.23:1515
INFO: Using agent name as: agent001
INFO: Send request to manager. Waiting for reply.
INFO: Received response with agent key
INFO: Valid key created. Finished.
INFO: Connection closed.
```

15. For further customization on `main.example.com`, shorten the interval that the **syscheck** runs. The default is for **syscheck** to run every 22 hours (79200 seconds). It may be helpful to reduce that time to 60 seconds during testing. Near the top of the file `/var/ossec/etc/ossec.conf` is a section `<syscheck>` with a `<frequency>` stanza that defaults to 79000 (or 79200) seconds, change that to 60 seconds and restart **ossec-hids**


```
# sed -i.bak -e  's/>79000</>60</ /var/ossec/etc/ossec.conf'
```

16. **OSSEC** does not report newly created files in directories. This feature can be added.

    - Turn on the "new files" feature by adding the stanza:

      `<alert_new_files>yes</alert_new_files>`

      after the frequency stanza.

      ```
      # sed -i.bak '/<\/frequency>$/ s:$:\n    <alert_new_files>yes<\/alert_new_files>:' /var/ossec/etc/ossec.conf
      ```

    - Now indicate to **syscheck** which directories to report the changes in real time by adding the `report_changes="yes"` and the `realtime="yes"` options to the directory entry in the `/var/ossec/etc/ossec.conf` file.

      Edit `/var/ossec/etc/ossec.conf` and locate the directory entries.

      ```
      <!-- Directories to check  (perform all possible verifications) -->
      <directories check_all="yes">/etc,/usr/bin,/usr/sbin</directories>
      <directories check_all="yes">/bin,/sbin</directories>
      <directories check_all="yes">/usr/local/bin</directories>
      ```

      Add the options:

      ```
      <!-- Directories to check  (perform all possible verifications) -->
      <directories report_changes="yes" realtime="yes" check_all="yes">/etc,/usr/bin,/usr/sbin</directories>
      <directories report_changes="yes" realtime="yes" check_all="yes">/bin,/sbin</directories>
      <directories report_changes="yes" realtime="yes" check_all="yes">/usr/local/bin</directories>
      ```

    - The addition of a new file is not normally an event of high enough priority to warrant sending an email but that can be overridden with a local rule.

      Locate and open in an editor the file `/var/ossec/rules/local_rules.xml`. Add before the closing tag the text below:

      ```
      <rule id="554" level="7" overwrite="yes">
        <category>ossec</category>
        <decoded_as>syscheck_new_entry</decoded_as>
        <description>File added to the system.</description>
        <group>syscheck,</group>
      </rule>
      ```

      The closing tag should look similar to:

      ```
      </group> <!-- SYSLOG,LOCAL -->
      ```

    - Restart ossec-hids.
    - Add a file to the monitored directory.
    - It may take some time, several minutes, (up to a couple of hours) before the email alert is sent. Monitor the `/var/ossec/logs/ossec.log` for **INFO: Ending syscheck scan.** File addition messages take longer to appear than file changes due to the way **OSSEC** works with its data structures.To see the alerts as they are processed watch the alerts log at `/var/ossec/logs/alerts/alerts.log`.

Chapter 7

Application Security

7.1 Labs

Lab 7

Exercise 7.1 Observing an **SUID** application

In this exercise we will observe the behavior of an **SUID** application without the suid bit set.

Make sure you start this exercise as a **normal user**.

First, find the `suid-app.c` application in the course solutions files. Start by taking a look at the source code and see if you understand what it does. Can you see how this application, given the right permissions, can be invoked by a regular user to escalate its privileges to the super-user?

Next we will compile it manually:

```
$ gcc suid-app.c -lpthread -o suid-app
```

You should now have access to a **suid-app** binary in the current directory. Execute it:

```
$ ./suid-app /etc/hosts
```

You should get a `setuid: Operation not permitted` error. This is due to the fact that the program is trying to escalate its privilege to root (UID 0) but the executable file's **SUID** bit is not set properly:

```
$ ls -l suid-app
-rwxr-xr-x 1 user user 9234 Aug  3 16:56 suid-app
```

Another way to figure out what is not working is to trace the application with **strace**. Try it and identify the line that precisely tells you why this application is not working as expected in the current context.

Lastly, try to correlate the output from **strace** with the application's source code.

Solution 7.1

What our **suid-app** execute does is to open any file as root, and dump its content on the terminal.

In the output from **strace**, you should see a line that looks as follows:

```
setuid(0) = -1 EPERM (Operation not permitted)
```

This indicates the application attempted a call to `setuid(0)`, but the system call failed since the operation was not permitted.

The **SUID** bit is obviously what is missing here.

Exercise 7.2 Applying the **SUID** bit

We saw in the previous exercise that an application that uses the `setuid()` call requires setting of the **SUID** bit on its executable in order to work as expected. In this exercise we will fix **suid-app** so that it works correctly.

As root, modify the permissions of **suid-app** so that the **SUID** bit is now enabled:

```
# chmod +s suid-app
```

You should now be able to see a new permission on that file:

```
$ ls -l suid-app
-rwsr-sr-x 1 user user 9234 Aug  3 16:56 suid-app
```

However, as you can see above, the owner of **suid-app** is still `user`. Let's change that to root: that way the **SUID** bit will take full effect.

```
# chown root:root suid-app
```

Take a look at **suid-app**'s permissions again. You should see that the **SUID** bit that was set previously is now cleared. What do you think happened?

Set the **SUID** bit again by using the `chmod` command. Make sure the **SUID** bit is set and that the owner of **suid-app** is `root`.

You should now be able to invoke the **suid-app** executable as a regular user:

```
$ whoami
user

$ ./suid-app /etc/lsb_release
thread id is 2954524416
This process' UID is 1000
This process' UID is now 0
DISTRIB_ID=LinuxMint
DISTRIB_RELEASE=14
DISTRIB_CODENAME=nadia
DISTRIB_DESCRIPTION="Linux Mint 14 Nadia"
```

Congratulations, you now have a working **SUID** application!

Solution 7.2

Whenever you change ownership of a **SUID** file, the kernel will, by policy, clear the **SUID** bit. This is why the **SUID** bit disappeared after setting ownership to `root`.

Exercise 7.3 Privileged operations with **SUID**

We will now execute a privileged operation as a regular user via our new **SUID** application.

As a regular user, try to access the `/etc/shadow` file. You should notice that this file is only readable by `root`:

```
$ cat /etc/shadow
$ cat: /etc/shadow: Permission denied

$ ls -l /etc/shadow
-rw-r----- 1 root shadow 1447 Aug  1 20:23 /etc/shadow
```


What our **suid-app** execute does is to open any file as root, and dump its content on the terminal. Execute this application on `/etc/shadow` and verify that you can read that file, as expected.

Make sure you keep the application available. We will reuse it in the next chapter.

Solution 7.3

Executing **suid-app** as a regular user should now let you access any file that is only readable as `root`:

```
$ ./suid-app /etc/shadow
thread id is 2461673216
This process' UID is 1000
This process' UID is now 0
root:$6$7kOePcai$ApJf/1DaJy2qePjBxu7zx.6sS4i03pQ6mJCPO944P9YPdLotMj7FzNvWJOi  \
                    hTgsQDKtFPrpDOkyj9K46azNaZ/:15789:0:99999:7:::
daemon:*:15630:0:99999:7:::
bin:*:15630:0:99999:7:::
sys:*:15630:0:99999:7:::
sync:*:15630:0:99999:7:::
games:*:15630:0:99999:7:::
man:*:15630:0:99999:7:::
lp:*:15630:0:99999:7:::
mail:*:15630:0:99999:7:::
news:*:15630:0:99999:7:::
uucp:*:15630:0:99999:7:::
proxy:*:15630:0:99999:7:::
...
```

Exercise 7.4 **SELinux** Protection

In this lab, we will explore the benefit of **SELinux** protection by installing an insecure web application and securing the system with **SELinux**. Our systems are already configured to run a web server and execute CGI scripts. With **SELinux** protection disabled, improperly written CGI scripts will be able to access parts of the system considered sensitive. By enabling **SELinux** policy enforcement, we will see how the kernel prevents the Apache web server from accessing those sensitive files.

In addition to being denied access to the sensitive data via the `password.sh` CGI script, there is an event driven daemon called **setroubleshoot-server** that will generate messages in `/var/log/messages` that will provide us with more information as to what happened.

The message provides an additional instruction to run the **sealert** command with a UUID that provides a significant amount of information about the **SELinux** policy violation that our script attempted. The information provided includes the process that attempted to violate the policy, the resource it attempted to access, what steps can be taken to allow that access and more. Be aware that allowing the access may NOT always be the right thing to do. In this case, the report is of a valid policy violation attempt.

- Confirm **SELinux** is disabled.
- Install and Verify the httpd webserver is running.
- Copy the `password.sh` script from the `SOLUTIONS` directory to `/var/www/cgi-bin/` directory and test.
- Enable **SELinux** in **permissive** mode.
- Re-boot the system to allow **SELinux** to relabel the files.
- Verify the status of **SELinux** is now **permissive**
- With **SELinux** in **permissive** mode verify the cgi script runs as before.(you may get a warning message pop-up,ignore it for now)
- Change **SELinux** to **enforcing** mode.
- Re-test the cgi-bin script, it should fail.
- Examine the recent contents of `/var/log/messages` for messages from **setroubleshoot**. One of the messages should suggest `"run sealert -l <long-UUID>`
- Run the **sealert** command from the previous step and examine the output.
- Set SELinux to **permissive** or **disabled**. Re-boot the system if setting **SELinux** to **disabled**.

Solution 7.4

- Confirm **SELinux** is disabled.

```
# getenforce
```

- Install and Verify the httpd webserver is running.

```
# yum -y install httpd elinks
# systemctl start httpd
# systemctl enable httpd
# elinks -dump http://main.example.com
```

- Copy the `password.sh` script from the `SOLUTIONS` directory to `/var/www/cgi-bin/` directory and test.

```
# elinks -dump http://main.example.com/cgi-bin/password.sh
```

- Enable **SELinux** in **permissive** mode.

```
# sed -i.bak -e 's/^SELINUX=.*/SELINUX=permissive/' /etc/selinux/config
```

- Re-boot the system to allow **SELinux** to relabel the files.

```
# reboot
```

- Verify the status of **SELinux** is now **permissive**.

```
# getenforce
```

- With **SELinux** in **permissive** mode verify the cgi script runs as before.(you may get a warning message pop-up,ignore it for now).

```
# elinks -dump http://main.example.com/cgi-bin/password.sh
```

- Change **SELinux** to **enforcing** mode.

```
# setenforce 1
```

- Re-test the cgi-bin script, **it should fail**.

```
# elinks -dump http://main.example.com/cgi-bin/password.sh
```

- Examine the recent contents of `/var/log/messages` for messages from **setroubleshoot**. One of the messages should suggest "`run sealert -l <ling-UUID>`

- Run the **sealert** command from the previous step and examine the output.

```
##### sample only #######
# run sealert -l 5fe766e3-1ec9-490e-8858-66f3942f2d2d
```

- Set SELinux to **permissive** or **disabled**. Re-boot the system if setting **SELinux** to **disabled**.

```
# setenforce 1
```

Chapter 8

Kernel Vulnerabilities

8.1 Labs

Lab 8

Exercise 8.1 Setting up **chroot**

In this exercise we will set up a chroot environment that can run the **bash** shell script interpreter. To do this you will recreate a minimal directory structure, and copy **bash** as well as its dependencies into it.

1. Use `/tmp/chroot-env/` as the top-level directory for this environment. Create this directory.
2. Determine all **bash** dependencies, running the **ldd** utility on `/bin/bash`.
3. Copy all required entries (including subdirectory structure) under `/tmp/chroot-env/`.
4. Test the **chroot** environment with:

   ```
   # chroot /tmp/chroot-env/ /bin/bash
   ```

You should get access to a shell, and not much else. You will notice that commands such as **ls** do not work. Why is that? How would you fix the problem?

Solution 8.1

1. Create the **chroot** directory:

   ```
   # mkdir /tmp/chroot-env/
   ```

2. Determine the dependencies:

   ```
   # ldd /bin/bash
           linux-vdso.so.1 =>  (0x00007fff509cc000)
           libtinfo.so.5 => /lib64/libtinfo.so.5 (0x00007f7f6502f000)
           libdl.so.2 => /lib64/libdl.so.2 (0x00007f7f64e2b000)
           libc.so.6 => /lib64/libc.so.6 (0x00007f7f64a69000)
           /lib64/ld-linux-x86-64.so.2 (0x0000560fee37c000)
   ```

   (Note: the exact file names and locations will depend on your **Linux** distribution and **bash** version.)

3. Recreate the directory structure that bash expects to find:

```
# mkdir /tmp/chroot-env/bin/
# mkdir /tmp/chroot-env/lib64/
```

Now copy the binary files:

```
# cp -L /bin/bash /tmp/chroot-env/bin/
# cd /lib64/
$ cp -L libtinfo.so.5 libdl.so.2 libc.so.6 ld-linux-x86-64.so.2 /tmp/chroot-env/lib64
```

(Note the use of the `-L` option to **cp** to de-reference symbolic links.)

4. You should now be able to invoke **chroot** (as root) and launch **bash** in its own **chroot** environment:

```
# chroot /tmp/chroot-env/ /bin/bash
```

Exercise 8.2 **suid-app** in a **chroot** environment

Identify the dependencies for the **suid-app** you built in the last chapter, and move it under the **chroot**'ed environment you just built.

Solution 8.2

- Copy the suid-app from the previous lab to `/tmp/chroot-env/bin/`. The **siud** bit(s) will need to be reset.
- Determine the dependencies for the **suid-app** and copy any required libraries to `/tmp/chroot-env/lib64`.

```
# ldd /tmp/chroot-env/bin/suid-app
        linux-vdso.so.1 =>  (0x00007fffc2bb3000)
        libpthread.so.0 => /lib64/libpthread.so.0 (0x00007f42e27c1000)
        libc.so.6 => /lib64/libc.so.6 (0x00007f42e23ff000)
        /lib64/ld-linux-x86-64.so.2 (0x00007f42e29f7000)
# cp -L /lib64/libpthread.so.0 /tmp/chroot-env/lib64/
```

- Copy a file to the chroot environment that only root may read.

```
# mkdir -p /tmp/chroot-env/etc/

# cp -L /etc/shadow /tmp/chroot-env/etc/
```

- Test the new application in the chrooted environment.

```
# chroot /tmp/chroot-env/ /bin/bash
bash-4.2# /bin/suid-app /etc/shadow
```

Exercise 8.3 Process Address Space Randomization

We will now experiment with the `kernel.randomize_va_space` **sysctl** and observe the effects this tunable item can have on regular processes. Use the user **student** as indicated during this exercise.

First identify the `kernel.randomize_va_space` tunable on your system. Feel free to use a virtual machine if `kernel.randomize_va_space` is not available with the kernel running on your system

Next, create a shell script that monitors prints the address of the heap, stack and vdso from the current process. This shell script could essentially contain:

```
$ grep -e heap -e vdso -e stack /proc/$$/maps
```

Test your shell script under the following three `kernel.randomize_va_space` settings, making sure to observe how the addresses for the stack, heap and vdso sections vary:

- 2 : Full address space randomization
- 1 : Partial randomization

- 0 : No randomization

Based on what you are seeing in the script's output, how do these three tunables differ?

Solution 8.3

To verify that `kernel.randomize_va_space` is available:

```
$ sudo  sysctl -a | grep randomize
     kernel.randomize_va_space = 2
```

Write a short script, `showstackaddr.sh`, containing:

```
#!/bin/bash
grep -e heap -e vdso -e stack /proc/$$/maps
```

Write another short script, say `runit.sh`, just to exercise it as required:

```
#!/bin/bash
echo -e "\nNow using randomize 2"
sudo sysctl kernel.randomize_va_space=2
./showstackaddr.sh
sudo sysctl kernel.randomize_va_space=2
./showstackaddr.sh
echo -e "\nNow using randomize 1"
sudo sysctl kernel.randomize_va_space=1
./showstackaddr.sh
sudo sysctl kernel.randomize_va_space=1
./showstackaddr.sh
echo -e "\nNow using randomize 0"
sudo sysctl kernel.randomize_va_space=0
./showstackaddr.sh
sudo sysctl kernel.randomize_va_space=0
./showstackaddr.sh
```

Then run it:

```
sudo chmod +x showstackaddr.sh runit.sh


[student@main ~]$ ./runit.sh

Now using randomize 2
kernel.randomize_va_space = 2
00db6000-00dd7000 rw-p 00000000 00:00 0                                  [heap]
7fff99e8c000-7fff99ead000 rw-p 00000000 00:00 0                          [stack]
7fff99edf000-7fff99ee1000 r-xp 00000000 00:00 0                          [vdso]
kernel.randomize_va_space = 2
00f42000-00f63000 rw-p 00000000 00:00 0                                  [heap]
7ffd26a23000-7ffd26a44000 rw-p 00000000 00:00 0                          [stack]
7ffd26a95000-7ffd26a97000 r-xp 00000000 00:00 0                          [vdso]

Now using randomize 1
kernel.randomize_va_space = 1
006e6000-0070d000 rw-p 00000000 00:00 0                                  [heap]
7ffdc1f86000-7ffdc1fa7000 rw-p 00000000 00:00 0                          [stack]
7ffdc1fbb000-7ffdc1fbd000 r-xp 00000000 00:00 0                          [vdso]
kernel.randomize_va_space = 1
006e6000-0070d000 rw-p 00000000 00:00 0                                  [heap]
7fffb514f000-7fffb5170000 rw-p 00000000 00:00 0                          [stack]
7fffb51f6000-7fffb51f8000 r-xp 00000000 00:00 0                          [vdso]
```

```
Now using randomize 0
kernel.randomize_va_space = 0
006e6000-0070d000 rw-p 00000000 00:00 0                                  [heap]
7ffff7ffa000-7ffff7ffc000 r-xp 00000000 00:00 0                          [vdso]
7fffffffde000-7ffffffff000 rw-p 00000000 00:00 0                         [stack]
kernel.randomize_va_space = 0
006e6000-0070d000 rw-p 00000000 00:00 0                                  [heap]
7ffff7ffa000-7ffff7ffc000 r-xp 00000000 00:00 0                          [vdso]
7fffffffde000-7ffffffff000 rw-p 00000000 00:00 0                         [stack]
```

As you can see, a value of 0 makes the address space of the process highly predictable: every invocation results in identical addresses.

A value of 2, on the other hand, result in full randomization: memory locations are determined when the process starts.

Exercise 8.4 Kernel pointers visibility [OPTIONAL]

Several rootkits and kernel exploits count on the fact that addresses of kernel functions are exposed via `/proc/kallsyms` and other files under `/proc`. Kernel developers added a feature known as `kptr_restrict` that blocks access to that information.

First determine if the kernel running on your system has this feature available by looking for the `kernel.kptr_restrict` **sysctl**:

```
# sysctl -a | grep kptr
```

Observe the difference in behavior when this sysctl is turned on (i.e. set to 1) versus when it is disabled (i.e. set to 0). Your test should involve printing the kernel symbols by looking at the content of `/proc/kallsyms` as a **regular user**.

Note that in addition to the `kernel.kptr_restrict` tunable, it is a good practice to hide kernel addresses from regular user processes too. Simply removing read permissions for the following two files is enough:

```
/boot/System.map-*
/boot/vmlinuz*
```

Solution 8.4 Setting `kernel.kptr_restrict` to 1 will replace all addresses with zeros, making it more difficult for attackers to figure out where specific kernel functions are located.

```
$ sudo sysctl  kernel.kptr_restrict
   kernel.kptr_restrict = 0

$ sudo sysctl  kernel.kptr_restrict=0

$ head /proc/kallsyms
0000000000000000 A irq_stack_union
0000000000000000 A __per_cpu_start
0000000000004000 A exception_stacks
0000000000009000 A gdt_page
000000000000a000 A espfix_waddr
000000000000a008 A espfix_stack
000000000000a020 A cpu_info
000000000000a0f0 A cpu_llc_shared_map
000000000000a0f8 A cpu_core_map
000000000000a100 A cpu_sibling_map

$ sudo sysctl  kernel.kptr_restrict=1

$ head /proc/ka0000000000000000 A irq_stack_union
0000000000000000 A __per_cpu_start
0000000000000000 A exception_stacks
0000000000000000 A gdt_page
0000000000000000 A espfix_waddr
```



```
0000000000000000 A espfix_stack
0000000000000000 A cpu_info
0000000000000000 A cpu_llc_shared_map
0000000000000000 A cpu_core_map
0000000000000000 A cpu_sibling_map
```

Exercise 8.5 Testing for kernel vulnerabilities, set up.

In the next few exercises we will use an older **Ubuntu 10.04** virtual machine to expose and test some kernel vulnerabilities. If you have not already done so, download the virtual machine image from the `training.linuxfoundation.org` web site as directed at the beginning of the course. Make sure it can be started as expected and that you can log into the virtual machine as `student` using the password `student`.

- Verify the modules needed for the exploit can be loaded into the kernel.
- Verify and correct the network adapter name to match the exploit.
- Compile the exploit programs.

Solution 8.5

- Verify the modules needed for the exploit can be loaded into the kernel.

  Verify modules can be loaded.

  ```
  $ sudo sysctl kernel.modules_disabled
  ```

  The **kernel.modules.disabled** should be **"0"** Also check the target kernel modules are not blacklisted, if so, comment them out.

  ```
  $ sudo grep -e rds -e econet /etc/modprobe.d/*

  $ sudo sed -i.bak -e 's/blacklist rds/#blacklist rds/' \
          -e 's/blacklist econet/#blacklist econet/' \
          /etc/modrobe.d/blacklist.conf
  ```

- Verify and correct the network adapter name to match the exploit.

  Before compiling our test programs, verify your network adapter name. The **full-nelson** was designed to use **eth0**. Virtual machine hypervisor may assign different network adapter types and **udev** will assign a new name for each adapter it finds remembering the old adapters at the same time. Verify your active network adapter is **eth0**, if not delete `/etc/udev/rules.d/70-persistent.rules` file and reboot.

  ```
  $ ifconfig -a
  $ sudo rm /etc/udev/rules.d/70-persistent.rules
  $ sudo reboot
  ```

- Compile the exploit programs. You should find two source files in the `student` home directory:

  ```
  full-nelson.c
  rds-fail.c
  $ gcc -o full-nelson full-nelson.c
  $ gcc -o rds-fail rds-fail.c
  ```

Exercise 8.6 RDS kernel vulnerability

The **Linux** kernel has the ability to load network protocol drivers on the fly when applications make use of that protocol. This is generally a good thing unless the code that is loaded cannot be trusted. This is exactly what happened with `CVE-2010-3904`: a user-space application running as a regular user opens a socket that uses the RDS protocol. The kernel then loads the corresponding kernel module (without any intervention from the super-user), which contains a known vulnerability.

This exploit is demonstrated in `rds-fail.c`. Feel free to take a look at the C code in order to familiarize yourself with the details.

Solution 8.6 Invoke the application as follows:


```
$ ./rds-fail

[*] Linux kernel >= 2.6.30 RDS socket exploit
[*] by Dan Rosenberg
[*] Resolving kernel addresses...
 [+] Resolved rds_proto_ops to 0xe09a9980
 [+] Resolved rds_ioctl to 0xe09a3090
 [+] Resolved commit_creds to 0xc0176140
 [+] Resolved prepare_kernel_cred to 0xc0176480
[*] Overwriting function pointer...
[...]
```

You should now have access to a shell running as `root`.

Reboot the virtual machine when you are done with this exercise.

Exercise 8.7 **econet** kernel vulnerability

The **econet** kernel module vulnerability was the result of an ancient driver (Econet, a network protocol used by British home computers from 1981) that is rarely used nowadays but still shipped with some distributions, including **Ubuntu 10.04**. Just like RDS, the **econet** module loads itself automatically when required. And just like RDS, the **econet** driver contains quite a few vulnerabilities. The exploit we will demonstrate makes use of three different vulnerabilities: CVE-2010-4258, CVE-2010-3849 and CVE-2010-3850.

Repeat the steps you executed in the last exercise, this time with the `full-nelson.c` application. Make sure the results are similar (i.e. when executing the application as a regular user, the application should eventually give you access to a `root` shell). You will need to open two shells, as the student user. Run **full-nelson** in the first shell to enable to vulnerability, then again in the second shell to activate it, thus elevating your privileges to `root`.

Reboot the virtual machine when you are done with this exercise. Use the freshly compiled **full-nelson** program.

- Run the **full-nelson** command, it will generate some output and appear to hang. This is normal.
- Switch to another console. (if using Virtual-Box press right-ctl + f2 keys)
- Run the **full-nelson** command on the second console. The program should give you a root shell.

Solution 8.7

Solutions are contained in the exercise instructions.

Exercise 8.8 Defeating kernel module exploits

We have seen in the last two exercises that the ability to load kernel modules on the fly constitutes a serious attack vector. We will now disable this functionality.

The first method we will leverage is the one that we discussed earlier in this chapter: just enable the following sysctl and repeat any of the exploits we tested previously:

```
$ sysctl kernel.modules_disabled=1
```

After executing the command above none of the exploits mentioned previously should be able to work.

Reboot the virtual machine when you are done with this exercise.

Exercise 8.9 Defeating kernel exploits by blacklisting modules

Removing the ability to load kernel modules dynamically is not always practical: some legitimate drivers may be needed at some point, which will require the corresponding kernel module to be loaded in memory. The `kernel.modules_disabled` **sysctl** is thus not always a valid option.

A more fine-grained approach consists in specifying which modules should never be loaded in the kernel. This method requires modifying (or adding) configuration files in `/etc/modprobe.d/`. We will use the second method here.

Open `/etc/modprobe.d/blacklist.conf` (or create it; some recent distributions such as **RHEL7** do not have this file by default) and add the following two lines at the end of the file:


```
blacklist rds
blacklist econet
```

Execute the **full-nelson** and **rds-fail** exploits again.

Since the **rds** and **econet** modules cannot be loaded, the code that is needed in order for the exploits to run correctly is just not present in memory.

Solution 8.9

Solutions are contained in the exercise instructions.

Chapter 9

Authentication

9.1 Labs

Lab 9

Exercise 9.1 Modifying Expiration and Improving Password Validation

In this exercise, the default password validation is improved by installing or customizing the `pam_pwquality` module.

The requirements are:

1. Confirm that the `pam_pwquality` module is installed.
2. Customize a new test user such that:
   - The test user cannot change their passwords for 3 days after updating the current password.
   - The test user's password must be changed every 30 days.
   - The user will begin receiving warnings about their password expiration 5 days prior to expiration.
   - The test user account will be locked 2 days after the password expires
3. Set the system defaults so these parameters are enforced for all new users.
4. Alter the password complexity such that:
   - Passwords must be a minimum of 12 characters in length.
   - Passwords must contain at least 3 character classes (upper-case, lower-case, digit or other)

Hint: Review the **man** pages for **useradd**, **chage**, **pam_pwquality** and **login.defs**.

Solution 9.1

1. Confirm that the `pam_pwquality` module is installed.

```
# find /usr -name \*pam_pwquality.so 2>/dev/null
# apt-get install libparm-pwquality
```

2. Customize a new test user such that:
   - The test user cannot change their passwords for 3 days after updating the current password.

- The test user's password must be changed every 30 days.
- The user will begin receiving warnings about their password expiration 5 days prior to expiration.
- The test user account will be locked 2 days after the password expires

```
# useradd testuser
# passwd -n 3 -x 30 -w 5 -i 2 testuser
```

3. To get all new users to have these expiration settings, edit `/etc/login.defs` as follows:

```
...
 # Password aging controls:
 #
 #      PASS_MAX_DAYS   Maximum number of days a password may be used.
 #      PASS_MIN_DAYS   Minimum number of days allowed between password changes.
 #      PASS_MIN_LEN    Minimum acceptable password length.
 #      PASS_WARN_AGE   Number of days warning given before a password expires.
 #
PASS_MAX_DAYS   30
PASS_MIN_DAYS   3
PASS_MIN_LEN    12
PASS_WARN_AGE   5
...
```

4. Alter the password complexity such that:

   - Passwords must be a minimum of 12 characters in length.
   - Passwords must contain at least 3 character classes (upper-case, lower-case, digit or other)
     In the `/etc/security/pwquality.conf` add/modify the following:

     ```
     minclass = 3
     minlen = 12
     ```

     Test the parameters by changing the test users password to **Linuxisgreat!**.

Exercise 9.2 Locking Accounts After Excessive Login Attempts

In this exercise, user accounts are to be locked after too many consecutive failed login attempts. To accomplish this, `pam_tally2` will be added to the authentication and account test stacks in the system wide PAM configuration file, `/etc/pam.d/password-auth` and `/etc/pam.d/system-auth` on **RHEL**-based systems `/etc/pam.d/common-auth` on **Ubuntu**-based systems.

1. Add a test user to the system.
2. Configure the system to reject any further login attempts even if the correct password is provided after 3 consecutive failed login attempts.
3. Attempt to login to the system with an invalid password more than the 3 times. Reviewing `/var/log/secure` will provide insight as to what is happening when the login attempts fail.
4. Reset the tally count by using the `pam_tally2` utility to allow the user to login to the system again.

Solution 9.2

1. Add a user to the system:

```
# useradd failuser
```

2. Configure the PAM file, `/etc/pam.d/system-auth /etc/pam/d/password-auth`, to reject too many invalid login attempts:

```
 #%PAM-1.0
 # This file is auto-generated.
 # User changes will be destroyed the next time authconfig is run.
auth        required      pam_env.so
auth        required      pam_tally2.so onerr=fail deny=3 unlock_time=60
auth        sufficient    pam_fprintd.so
```



```
auth        sufficient    pam_unix.so nullok try_first_pass
auth        requisite     pam_succeed_if.so uid >= 500 quiet
auth        required      pam_deny.so

account     required      pam_unix.so
account     sufficient    pam_localuser.so
account     sufficient    pam_succeed_if.so uid < 500 quiet
account     required      pam_permit.so
account     required      pam_tally2.so
...
```

3. Attempt to login to the system with an invalid password while running a **tailf** on `/var/log/secure` in another terminal window.

   Terminal 1:

   ```
   # ssh failuser@main.example.com
   ```

   Terminal 2:

   ```
   # tailf /var/log/secure
   ```

4. Once the maximum number of failed login attempts has been exceeded, reset the count to allow the user to try again.

   ```
   # pam_tally2 -u failuser
   # pam_tally2 -u failuser -r
   # pam_tally2 -u failuser
   ```

Exercise 9.3 Centralized Authentication using **LDAP** and **TLS** - Install an **LDAP server appliance**

This exercise will install a **LDAP** server appliance for centralized network authentication. The **turnkey openldap** appliance server was chosen for its availability on different distributions. Installing this appliance will create a new **VM**. The **appliance VM** is quite small, assigning only 512MiB of memory and using less than 1GB of disk space. The disk space is dynamically grow-able to 20GB but our usage remains quite low. There are many options to obtain the appliance and one option, the OVA image that is compatible with **VMWare** and **Virtual-Box** is available in the following directory.

`https://training.linuxfoundation.org/cm/LFS216`

Please see `https://www.turnkeylinux.org/openldap` for available options.

1. Import the appliance
2. Start the appliance
3. Configure the LDAP appliance on first boot with the following parameters:
   - `Root's password is  "pa$$word"`
   - `Admin's password id "pa$$w0rd"`
   - `The domain is "example.com"`
   - `No "HUB","System Notification" or "Update" services`

   This will create an LDAP server and present a summary screen of the the address and ports for the various servers available. Below is an example of the summary screen.

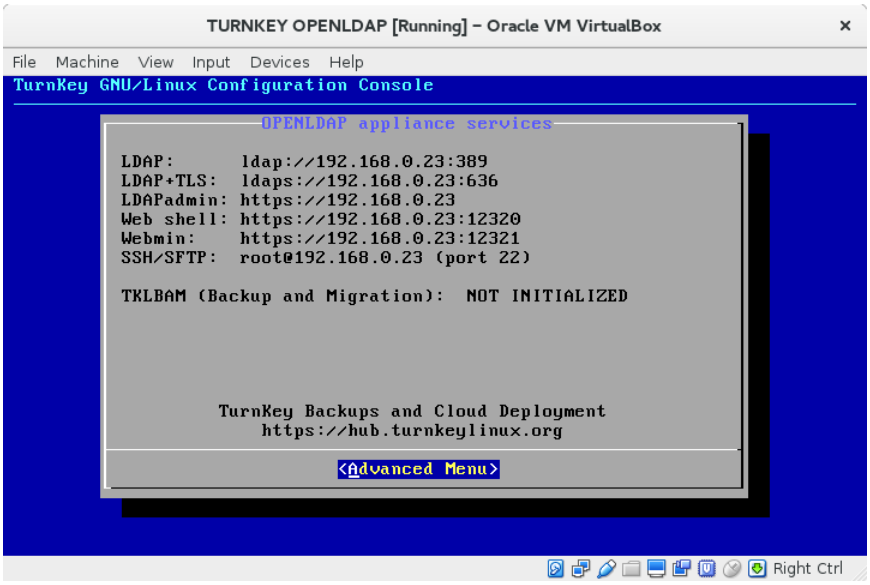

Figure 9.1: **Turnkey OpenLDAP Appliance Services**

Solution 9.3

1. Import the appliance
   - Verify the file `turnkey-openldap-14.1-jessie-amd64.ova` is available (version may be different).
   - Open the **Virtual-Box Manager**.
   - Select `File -> Import Appliance`
   - Enter location (or browse) of the **OpenLADP appliance**.
   - Select **Next** and continue as prompted.
2. Start the **TURNKEY OPENLDAP** virtual machine just created.
3. Configure the LDAP appliance on first boot with the following parameters:
   - `Root's password is "pa$$w0rd"`
   - `Admin's password id "pa$$w0rd"`
   - `The domain is "example.com"`
   - `No "HUB","System Notification" or "Update" services`

   The **Turnkey OpenLDAP Appliance Summary** should be visible, this is the normal **running** display of the appliance.

Exercise 9.4 Centralized Authentication using **LDAP** and **TLS** - testing the **LDAP** server and adding users.

In this exercise we Will verify the basic operation of the **LDAP** server. Specifically the ability of the server to respond to queries, add a **POSIX** group and user. Once the user is added we can monitor the connection with **wireshark** and observe the plain text transfer of information to and from the **LDAP** server. Since it is a poor security policy to have clear text user information, we will encrypt the data stream.

Note: This lab is going to use expired encryption keys, this is **NOT RECOMMENDED** in any sort of production environment. To accommodate our lab environment we will disable certificate verification. The ability to not verify the keys is handy in our lab but **DO NOT** use this technique on any production systems.

This exercise will be using the machine **secondary.example.com** to communicate with the **LDAP** server.

It is assumed that the `ready-for.sh` script has been run and the solutions file loaded and extracted to a directory.

1. Verify or install openldap-clients package

2. Using the **ip address** of the **LDAP** server displayed in the **Turnkey OpenLDAP Appliance Summary** screen in the previous exercise, perform a simple **LDAP** search of the **example.com** domain. We should see information about the base records for the domain example.com
3. Add a new group record using the `groups.ldif` file included in the **solutions** directory.
4. The **ldapsearch** command should now show a new group.
5. Add a new user record using the `users.ldif` file included in the **solutions** directory.
6. The **ldapsearch** command should now show a new user.
7. Install or verify wireshark-gnome is installed.
8. Capture an ldapsearch command with wireshark, notice the data is clear text.
9. Encrypt the **ldapsearch** command **this will fail due to "demo" ssl keys.**
10. Disable the verification of the certificate. **Note: the validity of the certificate is not checked anymore.**
    The encrypted ldapsearch should now work, confirm the data is encrypted in wireshark.
11. Re-try the encrypted **ldapsearch** command, it should now work and the data stream should be encrypted.

Solution 9.4

1. Verify or install openldap-clients package

```
# yum install openldap-clients sssd-tools
```

2. Using the **ip address** of the **LDAP** server displayed in the **Turnkey OpenLDAP Appliance Summary** screen in the previous exercise, perform a simple **LDAP** search of the **example.com** domain. We should see information about the base records for the domain example.com

```
# ldapsearch -x -H ldap://192.168.0.23 -b "dc=example,dc=com" -s sub"objectclass=*"
```

3. Add a new group record using the `groups.ldif` file included in the **solutions** directory.

```
# ldapadd -x -D "cn=admin,dc=example,dc=com" -W -H ldap://192.168.0.23 -f groups.ldif
```

4. The **ldapsearch** command should now show a new group.

```
# ldapsearch -x -H ldap://192.168.0.23 -b "dc=example,dc=com" -s sub"objectclass=*"
```

5. Add a new user record using the `users.ldif` file included in the **solutions** directory.

```
# ldapadd -x -D "cn=admin,dc=example,dc=com" -W -H ldap://192.168.0.23 -f users.ldif
```

6. The **ldapsearch** command should now show a new user.

```
# ldapsearch -x -H ldap://192.168.0.23 -b "dc=example,dc=com" -s sub"objectclass=*"
```

7. Install or verify wireshark-gnome is installed.

```
# yum install wireshark-gnome
```

8. Capture an ldapsearch command with wireshark, notice the data is clear text.

   Wireshark has excellent filters, use the filter `"(tcp.port==389) or (tcp.port==636)"` to have **wireshark** only display **LDAP** communication.

9. Encrypt the **ldapsearch** command **this will fail due to "demo" ssl keys.**

```
# ldapsearch -Z -x -H ldap://192.168.0.23 -b "dc=example,dc=com" -s sub"objectclass=*"
```

10. Disable the verification of the certificate.

    Note: the following is for a lab environment only.

```
# echo "TLS_REQCERT never" >> /etc/openldap/ldap.conf
```

Note: the validity of the certificate is not checked anymore. The encrypted ldapsearch should now work, confirm the data is encrypted in wireshark.

11. Re-try the encrypted **ldapsearch** command, it should now work and the data stream should be encrypted.

```
# ldapsearch -Z -x -H ldap://192.168.0.23 -b "dc=example,dc=com" -s sub"objectclass=*"
```

Exercise 9.5 Centralized Authentication using **LDAP** and **TLS** - User authentication

In this exercise, we explore what information is transmitted over the network when using **LDAP** for centralized authentication and how to secure that data stream by using TLS encryption.

1. Backup the original configuration.
2. Using the program `authconfig-tui`, configure the client, **secondary.example.com**, for **LDAP** authentication using the **Turnkey OpenLDAP** server previously installed with the following parameters:
   - Use **LDAP** for user Information.
   - Use **LDAP** for user Authentication.
   - The server is `ldap://<ip address of ldap server>`
   - **Base DN:** is **dc=example,dc=com**
3. Verify the user information is available for user **luser1**
4. Test the user authorization is functioning for user **luser1**
5. Verify the data stream is clear text
6. Enable **TLS** encryption for **LDAP**.
7. Disable the certificate validation. **Note: NOT RECOMMENDED in production systems.**
8. Test the user information is being supplied by **LDAP** for **luser1**
9. Test the user authorization is functioning for **luser1**.
10. Verify the data stream to and from the **LDAP** server is encrypted.
11. Restore the original configuration when completed.

Solution 9.5

1. Backup the original configuration.

```
# authconfig --savebackup=orig
```

2. Using the program `authconfig-tui`, configure the client, **secondary.example.com**, for **LDAP** authentication using the **Turnkey OpenLDAP** server previously installed with the following parameters:
   - Use **LDAP** for user Information.
   - Use **LDAP** for user Authentication.
   - The server is `ldap://<ip address of ldap server>`
   - **Base DN:** is **dc=example,dc=com**

```
# authconfig-tui
```


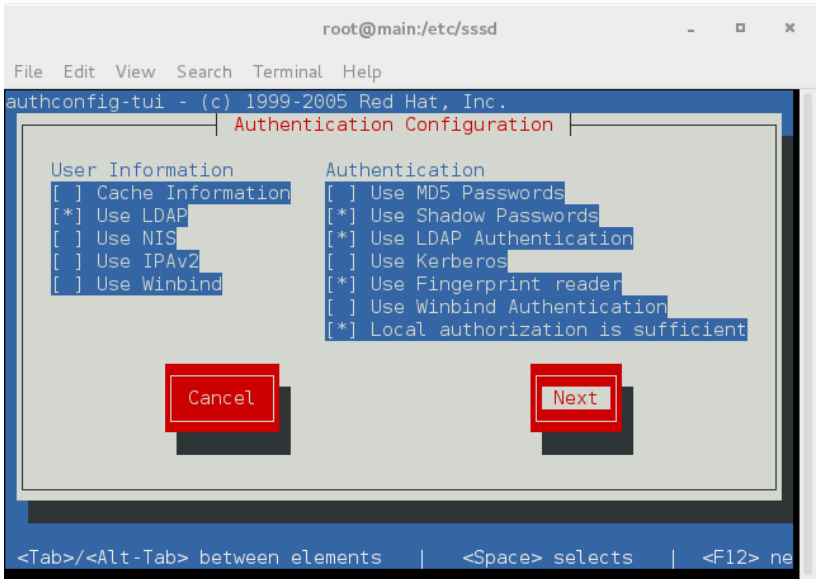

Figure 9.2: **Authentication Configuration - authconfig-tui screen 1**

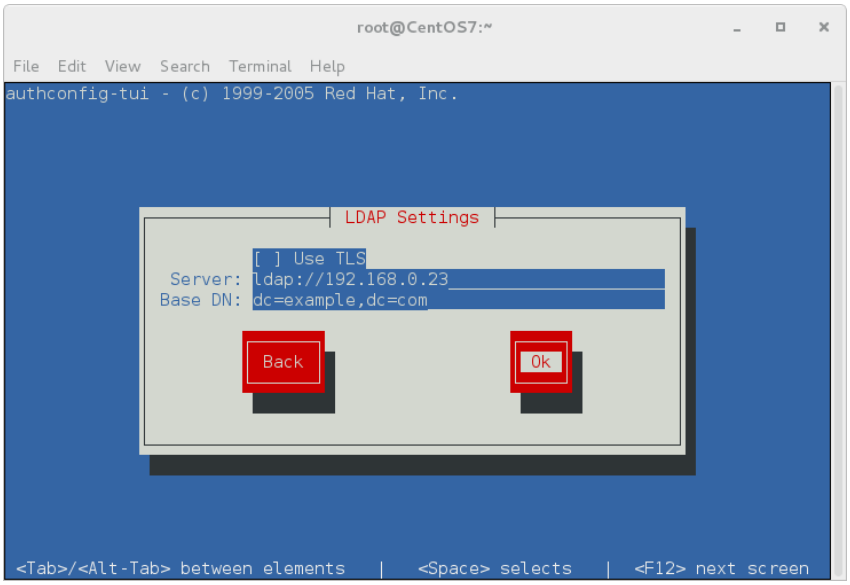

Figure 9.3: **Authentication Configuration - authconfig-tui screen 2 (no tls)**

3. Verify the user information is available for user **luser1**

   ```
   # getent passwd luser1
   ```

   The response should be:

   ```
   luser1:*:999001:999001:luser1:/home/users/luser1:
   ```

4. Test the user authorization is functioning for user **luser1**. The password should be password. There may not be a home directory available.

   ```
   # ssh luser1@localhost
   ```

5. Verify the data stream is clear text.

   - Open a **wireshark** session filtering on ports 389 and 636.

- While **wireshark** is capturing packets, log on and off as user luser1 via ssh to localhost.
- Observe the output in **wireshark**

6. Enable **TLS** encryption for **LDAP**. Run `authconfig-tui` and specify **TLS** for **LDAP**

```
# authconfig-tui
```

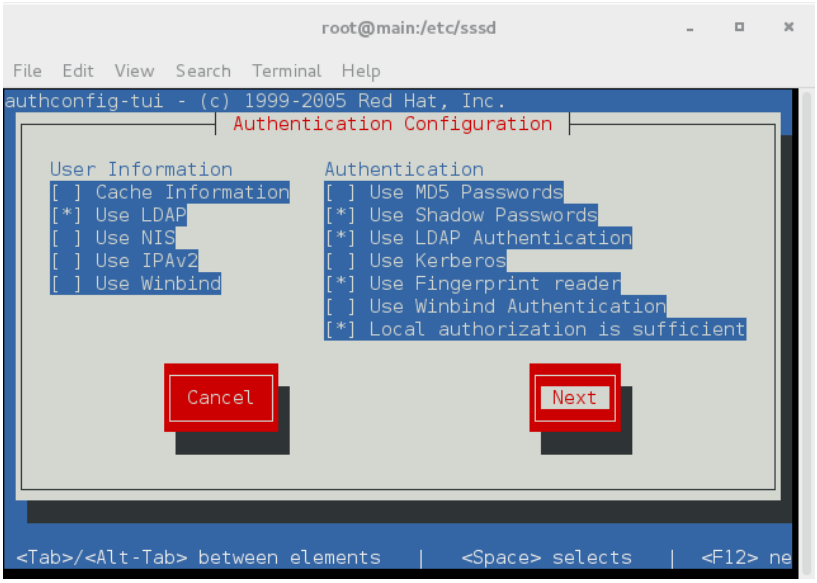

Figure 9.4: **Authentication Configuration - authconfig-tui screen 1 (tls)**

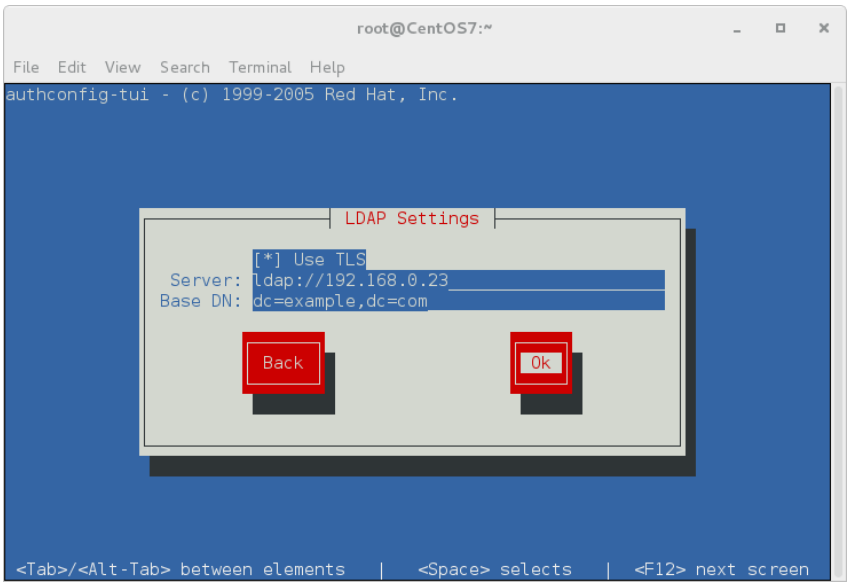

Figure 9.5: **Authentication Configuration - authconfig-tui screen 2 (tls)**

7. Disable the certificate validation. **Note: NOT RECOMMENDED in production systems.**
   Edit the `/etc/sssd/sssd.conf` file with your favorite editor. In the `[domain/default]` section add the line:
   `ldap_tls_reqcert = never`
   Restart the `sssd` service.

```
# systemctl restart sssd
```


8. Test the user information is being supplied by **LDAP** for **luser1**

```
# getent passwd luser1
```

9. Test the user authorization is functioning for **luser1**.

```
# ssh luser1@localhost
```

10. Verify the data stream to and from the **LDAP** server is encrypted.
    - Open a **wireshark** session filtering on ports 389 and 636.
    - While **wireshark** is capturing packets, log on and off as user luser1 via ssh to localhost.
    - Observe the output in **wireshark**.
11. Restore the original configuration when completed.

```
# authconfig --restorebackup=orig
```


Chapter 10

Local System Security

10.1 Labs

Lab 10

Exercise 10.1 Using **sudo** to Grant Elevated Execution Permissions

In this exercise we will show how to configure a user with specific sudo permissions. Many users will need root privileges to specific applications but not the entire system. To accomplish this you will want to set up **sudo** permissions for each user based on the application(s) they will need access to. The process for configuring **sudo** access is to create rules for each user in the `/etc/sudoers.conf` file or a separate file for each user in the `/etc/sudoers.d/` directory. To modify this file, you must use the **visudo** utility.

First, we will add a user and set a password for that user. Using **sudo** requires two configurations on the system:

1. A proper configuration in `/etc/sudoers`
2. Knowing the account password (likely since it is required for login).

```
# useradd testmysql
# echo "sudotest" | passwd --stdin testmysql
# visudo
```

The last command will drop you into an editor, **vi** in this case. The default configuration file has examples that can easily be modified to accomplish our required task. In this case, create a configuration that allows the `testmysql` user to execute /**bin**/**systemctl** for the **mariadb** service.

Next, switch to the `testmysql` account and test the sudo configuration by using /**sbin**/**service mysqld**.

```
$ su - testmysql
$ service mysqld start
$ sudo service mysqld start
$ sudo service mysqld restart
```

Now let's validate our configuration and make sure we have not allowed extra access. Try to start the **httpd** service.

```
$ sudo service httpd start
Error returned not allowing access to run.
exit
```

Bonus: Let's take advantage of the modular aspect of the /etc/sudoers configuration. We can create sudoers files with specific configurations in an include directory and leave the default configuration file as is.

```
# visudo -f /etc/sudoers.d/httpd
```

In this case, let's allow the testmysql user the ability to manage the **httpd** service.

Let's test our configuration:

```
$ su - testmysql
$ sudo service httpd start
$ sudo service httpd stop
```

Solution 10.1

First, we will add a user and set a password for that user. Using **sudo** requires two configurations on the system:

1. A proper configuration in /etc/sudoers
2. Knowing the account password (likely since it is required for login).

```
# useradd testmysql
# echo "sudotest" | passwd --stdin testmysql
# visudo
```

The last command will drop you into an editor, **vi** in this case. At end of the file add:

```
testmysql ALL=/sbin/service mysqld *
```

Next, switch to the testmysql account and test the **sudo** configuration by using /sbin/servicemysqld.

```
$ su - testmysql
$ service mysqld start
$ sudo service mysqld start
$ sudo service mysqld restart
```

Now let's validate our configuration and make sure we have not allowed extra access. Try to start the **httpd** service.

```
$ sudo service httpd start
Error returned not allowing access to run.
exit
```

Now, let's take advantage of the modular aspect of the /etc/sudoers configuration. We can create sudoers files with specific configurations in an include directory and leave the default configuration file as is.

```
# visudo -f /etc/sudoers.d/httpd
```

Add line in file:

```
testmysql ALL=/sbin/service httpd *
```

Let's test our configuration:

```
$ su - testmysql
$ sudo service httpd start
$ sudo service httpd stop
```


Exercise 10.2 Filesystem ACLs and What Are They Good For

Filesystem ACLs allow users to grant permissions to the files they own to other users on the system without the need to have a admin user take any action. This is most useful for collaborating with users on the system outside of the formally declared system groups as in `/etc/group`. Filesystem ACLs can also be used to restrict access to files that a user might normally have access to given its **UNIX** permissions.

In order to use filesystem ACLs, the filesystem must support and have the `acl` mount option enabled. This mount option can be set in two places on **Linux** systems:

1. `/etc/fstab` in column 4 for mount options
2. in the filesystem superblock in the `Default mount options` section, viewable with

   ```
   # tune2fs -l FILESYSTEM-DEV
   ```

   where `FILESYSTEM-DEV` can be `/dev/sda1`, etc.

Filesystems created by **anaconda** at install time have the `acl` option automatically added to the filesystem superblock.

Review the **man** page for `setfacl` and `getfacl`, as there are explicit examples in those pages that can be used to accomplish most ACL related tasks.

In this lab exercise, we are going to create some users, a shared directory and apply file-sharing permissions. We will then apply specific permissions to the files.

- First, create a collaborative directory `/shared/mutants` with the group ownership of a new group **heroes**.
- Add three users, **sylar** and **cheerleader** that have a supplementary group of **heroes'**. Create another user **sue** that does not belong to the group **heroes**.
- Do not forget to set the **SUID** bit on the directory.
- Switch to the user **sylar** and **cheerleader** and observe how each can create and edit files in the `/shared/mutants` directory, regardless of which one created the file.
- Verify that user **sue** has no access to the directory `/shared/heroes` and its files.
- Add an ACL to allow **sue** access to create,read and modify files in the directory `/shared/heroes`.
- As user **cheerleader** create a file called `/shared/mutants/private` and explicitly deny access to it from the user **sylar** using an ACL.

Solution 10.2

- First, create a collaborative directory `/shared/mutants` with the group ownership of a new group **heroes**.

  ```
  # groupadd -r heroes
  # mkdir -p /shared/mutants
  # chgrp heroes /shared/mutants
  ```

- Add three users, **sylar** and **cheerleader** that have a supplementary group of **heroes**. Create another user **sue** that does not belong to the group **heroes**.

  ```
  # useradd sylar
  # useradd cheerleader
  # useradd sue
  # usermod -aG heroes sylar
  # usermod -aG heroes cheerleader
  ```

- Do not forget to set the **SUID** bit on the directory.

  ```
  # chmod g+ws,o=- /shared/mutants
  ```

- Switch to the user **sylar** and **cheerleader** and observe how each can create and edit files in the `/shared/mutants` directory, regardless of which one created the file.


```
# su - sylar
$  echo "Precognition" >> /shared/mutants/victims
$ exit
# su - cheerleader
$ echo "\nI will stop you." >> /shared/mutants/victims
$ exit
```

- Verify that user **sue** has no access to the directory `/shared/heroes` and its files.

```
# su - sue
$ cat /shared/mutants/victims

        \item Add an ACL to allow \textbf{sue} access to create,read and modify files in
        the directory \filelink{/shared/heroes}.Set two ACLs one for directory access
        and the other is a default on the directory for new files.
        \begin{raw}
# setfacl -m d:u:sue:rwx /shared/mutants
# setfacl -m u:sue:rwx /shared/mutants
```

  To access existing files, the ACLs will need to be adjusted manually.

```
# setfacl -m u:sue:rw /shared/mutants/victims
```

- As user **cheerleader** create a file called `/shared/mutants/private` and explicitly deny access to it from the user **sylar** using an ACL.

```
# su - cheerleader
$ echo "private stuff" >> /shared/mutants/private
$ setfacl -m u:sylar:- /shared/mutants/private
$ getfacl  /shared/mutants/private
$ exit

# su - sylar
$ cat /shared/mutants/private

cat: /shared/mutants/private: Permission denied

$ exit
```

Exercise 10.3 File Attributes, protecting of Critical Files

The Advanced Intrusion Detection Environment (**AIDE**) tool was mentioned in the Auditing and Detection unit. **AIDE** is a post-event detection utility, so it can only report what has already happened. File attributes provide additional restrictions on files and may prevent some files from being changed.

There are two commands **lsattr** and **chattr** that are used to display and change the attributes. The `man chattr` describes the options and related restrictions. Listed below are a couple of interesting **chattr** options.

- `lsattr <file>` to display the file attributes
- `chattr +i <file>` adds the immutable option
- `chattr +a <file>` adds the append only option

Create a file and test the **i** and **a** options of **chattr**.

Solution 10.3

The following are examples of the **chattr** command.

- Create a new file and check the `file attributes`

```
$ touch foo-file
$ lsattr foo-file
```


The default `file attributes` should be a shown below.

```
-------------e-- foo-file
```

- Check that you can append data to the file.

```
$ echo "stuff" >> foo-file
```

- Set the append option on the file and test.

```
$ sudo chattr +a foo-file
$ lsattr foo-file
```

The output of **lsatter** should show the append option as active.

```
-----a-------e-- foo-file
```

- Now try to overwrite the file, it should fail.

```
$ echo "stuff" > foo-file
bash: foo-file: Operation not permitted
```

- The append to the file should work.

```
$ echo "stuff" >> foo-file
```

- Set and test the `immutable` option.

```
$ sudo chattr +i foo-file
$ echo "stuff" >> foo-file
bash: foo-file: Permission denied
$ sudo echo "stuff" >> foo-file
bash: foo-file: Permission denied
```

- Undo the options.

```
$ sudo chattr -i -a foo-file
```


Chapter 11

Network Security

11.1 Labs

Lab 11

Exercise 11.1 Stateful Packet Filtering

In this exercise, we will explore **stateful** vs **non-stateful** packet filtering using the **Linux** firewall code, **netfilter**. To prepare for this exercise, review the **man** page for `iptables-extensions`. Search for the `conntrack` and `--state` keywords.

Earlier we mentioned ingress and egress filtering and here we will see how using state tracking (stateful) firewall configurations can simplify rulesets by allowing us to create the complex ruleset on one side of the firewall and a single rule for the opposite side of the firewall.

This lab exercise will be done on the `secondary.example.com` system and tested with the `main.example.com` system. Login to the two systems using the console, NOT the SSH service: **This is CRITICAL for this exercise.** Next, run **tcpdump** or **wireshark** on both systems in order to see what traffic is allowed and what is not.

Note: This exercise assumes no other program is modifying **netfilter's** tables. Stop any other firewall control programs **UFW**, **iptables**, **firewalld** etc.

1. Start by adding a `REJECT` rule to both the `INPUT` and `OUTPUT` chains.
2. Attempt to **ssh** to `main.example.com`. What is wrong?
3. Insert a rule to the `OUTPUT` that allows SSH traffic to exit the system.
4. Again, attempt to **ssh** to `main.example.com`.
5. Insert a rule that allows SSH traffic through the `INPUT` chain.
6. Let's try something simpler: **ping** the `main.example.com` system. What happens?
7. Insert an allow rule to the `OUTPUT` chain to allow the ICMP packets out.
8. **ping** `main.example.com` again. What happens now?

Instead of adding another allow rule to the `INPUT` chain in order to get the **ping** replies, let's simplify our ruleset.

1. Remove the SSH allow rule on the `INPUT` chain.
2. Insert a rule that matches on states `ESTABLISHED` and **RELATED** to the `INPUT` chain.

ping and **ssh** both now be allowed.

Solution 11.1

1. It is a good idea to allow loopback traffic first.

```
# iptables -A INPUT -i lo -j ACCEPT
# iptables -A INPUT -j REJECT
# iptables -A OUTPUT -j REJECT
```

2. `$ ssh main.example.com`

   Traffic is not allowed to exit the system: it is blocked at the `OUTPUT` chain.

3. `# iptables -I OUTPUT -p tcp --dport 22 -j ACCEPT`

4. `$ ssh main.example.com`

   Still blocked. This time the `INPUT` chain is blocking the responses from the `main.example.com` server.

5. `# iptables -I INPUT -p tcp --sport 22 -j ACCEPT`

6. `$ ping main.example.com`

   ping packets cannot leave the system. The `OUTPUT` reject rule is blocking packets from leaving.

7. `# iptables -I OUTPUT -p icmp -j ACCEPT`

8. `$ ping main.example.com`

   Again, the `INPUT` chain `REJECT` rule is blocking traffic from getting back to our **ping** command.

Instead of adding another allow rule to the INPUT chain in order to get the ping replies, let's simplify our ruleset.

```
# iptables -D INPUT -p tcp --sport 22 -j ACCEPT
```

1. `# iptables -I INPUT -m state --state ESTABLISHED,RELATED -j ACCEPT`

Now **ping** and **ssh** should both be allowed.

Exercise 11.2 Network-related kernel tunables

The kernel offer a number of tunables (via the **sysctl** interface) that are designed to make the system more secure. Note that some of the the following tunables may not be available on certain kernel versions or distribution-configured kernels. Let's now go through some of them:

Some multi-homed networks or complex routing configurations allow for incoming traffic to flow through one interface and outgoing traffic to flow through another. For environments where that is not the case, the kernel makes it possible to check the routing table against the source address of incoming packets to make sure that they are coming from the interface our routing table says that address is on. This is called **source route verification**. Enable it on your system with:

```
# sysctl net.ipv4.conf.all.rp_filter=1
```

A very common denial of service attack consists in a system sending several SYN packets to your server without completing the TCP three-step handshake. This can prevent the kernel from servicing legitimate connections. Apply the follow tunable to defeat SYN flood attacks:

```
# sysctl net.ipv4.tcp_syncookies=1
```

The ICMP protocol implementation in the **Linux** kernel can be adjusted based on your security requirements. Disable ICMP echo replies and ICMP broadcast replies with the following tunables:


```
# sysctl net.ipv4.icmp.echo_ignore_all=1
# sysctl net.ipv4.icmp.echo_ignore_broadcasts=1
```

Moreover, the maximum rate at which the kernel generates icmp messages of the types specified by `icmp_ratemask` can be set in `icmp_ratelimit`. The value is the number of jiffies (usually milliseconds) the kernel has to wait between sending two such messages. Set `icmp_ratelimit` to an interval of 100 jiffies by doing:

```
# sysctl net.ipv4.icmp_ratelimit=100
```

Chapter 12

Network Services Security

12.1 Labs

Lab 12

Exercise 12.1 Using **Netcat** to Clone Hard Drives over the Network

This lab is based on the example from this web page: `http://www.howtoforge.com/useful-uses-of-netcat`

We will copy a partition (you can copy a whole disk too) over the network from a target system to a host, where we will analyze the image at some later point in time. To accomplish this, it is probably best to boot from **trusted media** (such as we will discuss in our lab exercise in the **Response Mitigation** session) and then perform this network copy.

Keep in mind, this is an emergency method for getting data across the network without encryption or authentication.

Note: Your system may use a different hard drive device node than the one used in the below example. Use `sudo fdisk -l` to determine the hard drive device file on your system. Use that device file in place of `/dev/sda` in the exercise.

On `main.example.com`, start the **netcat** listener and pipe the output to an image file.

```
# nc -l 1234 | dd of=/tmp/secondary-boot-partition.img
```

On `secondary.example.com`, start the copy of the /boot partition to `main.example.com` using **netcat**:

```
# dd if=/dev/sda1 | nc main.example.com 1234
```

Exercise 12.2 Using **Netcat** to Spoof HTTP Headers

This lab is based on the example from this web page: `http://www.howtoforge.com/useful-uses-of-netcat`

Sometimes it is desirable or necessary to change the headers in HTTP requests to remote web servers. Changing headers with unique identifiers can add a measure of privacy to web requests, testing responses to different User-Agents or specific headers, or to gain access to specific data that the web service is configured to provide only in response to requests with specific headers are all potential reasons for HTTP Header Spoofing. The **Firefox** web browser extension *User Agent Switcher* does this from a drop down menu in the **Firefox** interface.

From `secondary.example.com`, you can use **netcat** to request web pages:

```
# nc main.example.com 80
```

You can then type in headers as follows:

```
GET / HTTP/1.1
Host: node.example.com
Referrer: foo-example.com
User-Agent: test-browser
```

At this point, you must hit enter twice in order to send the EOF character to the web server and it will respond with the page that you have requested, in this case the default index page. The *User Agent Switcher* extension in **Firefox** does something very similar to this to fake remote web sites into thinking the browser (as self-described by the User-Agent header) is a browser other than **Firefox**.

There are well defined HTTP headers as defined in the RFC that you can send. Additionally, a web site operator can also look for custom ones if they also control the application that is used to access that web service.

Many smart phone applications are really requesting data from web services, often using a custom header sent by the smart phone app.

Here is a complete example:

```
secondary:~# nc main.example.com 80
GET / HTTP/1.1
Host: foo.example.com
Referrer: secondary.example.com
User-Agent: test-browser

HTTP/1.1 200 OK
Date: Thu, 10 Oct 2013 20:26:10 GMT
Server: Apache/2.2.15 (CentOS)
Last-Modified: Thu, 10 Oct 2013 20:26:02 GMT
ETag: "fc6a-5-4e868cf7d64d7"
Accept-Ranges: bytes
Content-Length: 5
Connection: close
Content-Type: text/html; charset=UTF-8
[...]
```

Look in `/var/log/httpd/access_log` to see what UserAgent is being logged.

Exercise 12.3 Allowing **NFS** access through a Firewall

First of all, use NFSv4. The clients and server will negotiate the highest NFS protocol version that both are capable of using. Version 4 also has significant security and management benefits over previous versions. Since NFS is an RPC (Remote Procedure Call) service, it will be necessary to allow clients to access TCP and UDP port 111.

The following workflow will accomplish the task of configuring an NFS service through a **Linux** firewall:

1. Add firewall rules for client access to the NFS service.
2. Create a directory to share, an `export` directory.
3. Configure the `/etc/exports` file. See the **man** page, `man 5 exports`.
4. Restart the **portmapper**, **nfs**, and **nfslock** services.

If you have to support older version NFS clients, then do the following:

1. Configure NFS daemons to listen on specific ports - edit `/etc/sysconfig/nfs` and set the `PORT` variables.
2. Add firewall rules for the ports configured in the `/etc/sysconfig/nfs` file.
3. Then follow the steps above.

Solution 12.3

1. ```
   # iptables -I INPUT -p udp --dport 2049 -j ACCEPT
   # iptables -I INPUT -p tcp --dport 2049 -j ACCEPT
   # iptables -I INPUT -p udp --dport 111 -j ACCEPT
   # iptables -I INPUT -p tcp --dport 111 -j ACCEPT
   ```


   ```

2. `# mkdir /shared/export`

3. `# echo "/shared/export        *.example.com(ro)" >> /etc/exports`

4. ```
   # for i in rpcbind nfslock nfs
   do
           systemctl restart  $i
   done
   ```

If you have to support older version NFS clients, then do the following:

1. ```
   # grep PORT /etc/sysconfig/nfs
   # sed -i.bak -e 's/#LOCKD_TCPPORT=.*/LOCKD_TCPPORT=4001/'
     -e 's/#LOCKD_UDPPORT=.*/LOCKD_UDPPORT=4002/'
     -e 's/#RQUOTAD_PORT=.*/RQUOTAD_PORT=4003/'
     -e 's/#MOUNTD_PORT=.*/MOUNTD_PORT=4004/'
     -e 's/#STATD_PORT=.*/STATD_PORT=4005/'
     -e 's/#STATD_OUTGOING_PORT=2020/#STATD_OUTGOING_PORT=4006/'
   # iptables -I INPUT -p udp --dport 4001:4005 -j ACCEPT
   # iptables -I INPUT -p tcp --dport 4001:4005 -j ACCEPT
   # iptables -I OUTPUT -p udp --sport 4006 -j ACCEPT
   ```

2. ```
   # grep PORT /etc/sysconfig/nfs
   # sed -i.bak -e 's/#LOCKD_TCPPORT=.*/LOCKD_TCPPORT=4001/'
     -e 's/#LOCKD_UDPPORT=.*/LOCKD_UDPPORT=4002/'
     -e 's/#RQUOTAD_PORT=.*/RQUOTAD_PORT=4003/'
     -e 's/#MOUNTD_PORT=.*/MOUNTD_PORT=4004/'
     -e 's/#STATD_PORT=.*/STATD_PORT=4005/'
     -e 's/#STATD_OUTGOING_PORT=2020/#STATD_OUTGOING_PORT=4006/'
   ```

3. ```
   # iptables -I INPUT -p udp --dport 4001:4005 -j ACCEPT
   # iptables -I INPUT -p tcp --dport 4001:4005 -j ACCEPT
   # iptables -I OUTPUT -p udp --sport 4006 -j ACCEPT
   ```

4. Then follow the steps for NFSv4.

Chapter 13

Denial of Service

13.1 Labs

Lab 13

Exercise 13.1 **ping** Flood Attack

In this exercise, we are going to initiate a **ping** flood from `main.example.com` against `secondary.example.com`. Using **tcpdump**, we will construct a command line that displays the packet attack in real-time. Review the `man pcap-filter` page for information on how to construct a **tcpdump** and **wireshark** filter.

1. Open 4 terminal windows on `main.example.com`.
2. Open a single terminal on `secondary.example.com`.
3. On `secondary.example.com`, construct a **tcpdump** command that displays ICMP traffic only.
4. Start **ping** with the flag that indicates a ping flood from the each terminal on `main.example.com`.
5. Observe what happens in the **tcpdump** terminal
6. Use `CTRL-c` to terminate the **ping** flood commands in all 4 terminal windows on `main.example.com`.
7. What methods could we construct to mitigate this type of attack?
8. In the firewalling unit we will learn about using the `RECENT` module to rate limit specific types of traffic through Netfilter.

Solution 13.1

3. This will need to be done as root or with **sudo**. The **man** page for **pcap-filter** indicates that using `icmp` is enough filter the packets to simply show us ICMP packets.

   ```
   # tcpdump icmp
   ```

4. ```
   # ping -f secondary.example.com
   ```

7. As we will discuss in a later chapter, using the **Netfilter** `RECENT` module and a ruleset as follows will rate limit the number of ICMP packets we reply to:
   ```

```
# iptables -A INPUT -i eth0 -p icmp -m recent --set
# iptables -A INPUT -i eth0 -p icmp -m recent --update --seconds 60 --hitcount 60 -j REJECT
# iptables -A INPUT -i eth0 -p icmp -j ACCEPT
# iptables -A INPUT -i eth0 -j REJECT --reject-with=icmp-host-unreachable
```

Chapter 14

Remote Access

14.1 Labs

Lab 14

Exercise 14.1 Unencrypted Network Sessions:

Here we will demonstrate why any remote communication that includes authentication/authorization data (or other sensitive information) needs to be protected via encrypted network communication channels.

Using the **telnet** utility, we will connect to a remote system and observe network communications during a simple unencrypted network session.

- Install and test the **telnet** server on the **secondary.example.com** system.
- Trace the telnet login process from **main.example.com** to **secondary.example.com** using wireshark.

Solution 14.1

1. Ensure that the **telnet** server is installed on **secondary.example.com**:
   - On **CentOS**:
     ```
     # yum install telnet-server
     ```
   - On **OpenSUSE**:
     ```
     # zypper install telnet-server
     ```
   - On **Debian**:
     ```
     # apt-get install telnetd xinetd
     ```
2. Enable and test the **telnet** server: NOTE: you may have to enable the **xinetd** daemon.
   - On **CentOS** or **OpenSUSE**
     ```
     #  systemctl start telnet.socket
          #  telnet localhost
     ```

- On **Debian**:

  Create an **xinetd** configuration file for the **telnet** daemon. Edit the file `/etc/xinetd.d/telnet` and add the following contents:

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

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

```
#  systemctl restart xinetd
```

- Verify the telnet session is functioning from **main.example.com** to **secondary.example.com**:

```
[main]# telnet secondary.example.com
```

  Note: The firewall may require adjusting or terminating on **secondary.example.com**.

```
# iptables -I INPUT -p tcp --dport 23 -j ACCEPT
```

  or for **firewalld** check the active **zone**, then add a temporary rule.

```
# firewall-cmd --get-active-zones
# firewall-cmd --zone=public --add-port=23/tcp
```

3. Using **wireshark**, open a capture session from **main.example.com** on the adapter associated with **main.example.com** (the private network) and trace the login process from **main.example.com** to **secondary.example.com**.

   You will be prompted for credentials, use the `student` account to login. After performing the login steps, observe the traffic that was captured by **wireshark**. (Hint: if you right-click on the initial packet of the session, you can select `Follow TCP Stream`, and the entire session will be reassembled and rendered in a human readable format.)

Using **wireshark** (which allows the viewing of network traffic packets on the wire), we intercepted and re-assembled the **telnet** session, thus allowing us to view all communications between both hosts. If this communication were encrypted, the session would have appeared as meaningless ASCII characters, or random bytes; but as an unencrypted **telnet** session (in this case), we see the communication(s) as a line-by-line ***wire-tap*** transcript.

Exercise 14.2 Configuring an OpenVPN Service and Clients

In this exercise we sill set up **main.example.com** as our **VPN Server** and **secondary.example.com** as our **VPN Client**. The following **URL** may be used as a guide:
`https://openvpn.net/index.php/open-source/documentation/miscellaneous/78-static-key-mini-howto.html`.

Additional configuration examples may be found at:

`/usr/share/doc/openvpn-<version>/sample/sample/` directory.

Sample configurations for this exercise are included in the appropriate **Solutions** directory.

Confirm **openvpn** is installed before continuing:

```
# yum install openvpn
```

1. On **main.example.com**:
   - Open the appropriate firewall port on **main.example.com**.
   - Create a static shared key.
   - Create the configuration file for **main.example.com** and place it in the file `/etc/openvpn/main.conf`.
   - Start the server side of the **VPN**.
2. On **secondary.example.com**:
   - Copy the static shared key from **main.example.com** to **secondary.example.com**.

- Create the configuration file for **secondary.example.com** and place it in the file `/etc/openvpn/secondary.conf`.
- Start the client side of the **VPN**.

3. Test the tunnel by using **telnet** and **wireshark**.

Solution 14.2

1. On **main.example.com**:
   - Open the appropriate firewall port on **main.example.com**.

     ```
     # iptables -I INPUT -p udp  --dport 1194 -j ACCEPT
     ```

     or for **firewalld** check the active **zone**, then add a temporary rule.

     ```
     # firewall-cmd --get-active-zones
     # firewall-cmd --zone=public --add-port=1194/udp
     ```

   - Create a static shared key.

     ```
     $ openvpn --genkey --secret /etc/openvpn/static.key
     ```

   - Create the configuration file for **main.example.com** and place it in the file `/etc/openvpn/main.conf`.

     ```
     dev tun
     ifconfig 10.8.0.1 10.8.0.2
     secret static.key
     ```

   - Start the server side of the **VPN**.

     ```
     # systemctl start openvpn@main
     ```

2. On **secondary.example.com**:
   - Copy the static shared key from **main.example.com** to **secondary.example.com**.

     ```
     # scp main.example.com:/etc/openvpn/static.key /etc/openvpn/
     ```

   - Create the configuration file for **secondary.example.com** and place it in the file `/etc/openvpn/secondary.conf`.

     ```
     remote main.example.com
     dev tun
     ifconfig 10.8.0.2 10.8.0.1
     secret static.key
     ```

   - Start the client side of the **VPN**.

     ```
     # systemctl start openvpn@secondary
     ```

3. Test the tunnel by using **telnet** and **wireshark**.

   Note: The sequence is important as the remote `<secondary>` must contact the server `<main>` to establish the connection. Once started the **VPN** is bidirectional. The most common challenges are firewall related.

   From the **secondary.example.com**:

   ```
   # ping 10.8.0.1
   ```

   From the **main.example.com**:

   ```
   # ping 10.8.0.2
   ```

   From the **main.example.com**:

   ```
   # telnet 10.8.0.2
   ```

Chapter 15

Firewalling and Packet Filtering

15.1 Labs

Lab 15

Exercise 15.1 **Netfilter** (**iptables**) Rate Limiting with the **recent** Module

Netfilter provides a **recent** module that can dynamically create deny or allow rules based on how many times a specific type of traffic reaches the system. More information on this module can be found in the **man** page for **iptables-extensions**.

In this exercise, we are going to use the `recent` module to create a ruleset that will significantly reduce the number of brute force SSH login attempts by dynamically blocking IP addresses that create too many `NEW` connections to the SSH daemon.

Let's begin by resetting our ruleset to an open firewall. The `-P` flag will reset the default policy for the chain, in thise case to `ACCEPT`, and the `-F` flag will remove all rules from the chain:

```
# iptables -P INPUT ACCEPT
# iptables -P OUTPUT ACCEPT
# iptables -F
```

We should always have the following as starting point for our **Netfilter** firewalls: allow traffic on the loopback interface and allow established and related traffic into the system:

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

Next, let's add the following rules which will dynamically add `DROP` rules for IP addresses that exceed 2 `NEW` TCP connection attempts in a 60 second time frame:

```
# iptables -A INPUT -i eth0 -p tcp --dport 22 -m state --state NEW -m recent --set
# iptables -A INPUT -i eth0 -p tcp --dport 22 -m state --state NEW -m recent \
          --update --seconds 60 --hitcount 2 -j DROP
# iptables -A INPUT -i eth0 -p tcp --dport 22 -j ACCEPT
# iptables -A INPUT -i eth0 -j REJECT --reject-with=icmp-host-unreachable
```

Let's test our configuration - from `secondary.example.com`, attempt to **ssh** into the system more than 2 times using an incorrect password. Keep in mind, the **ssh** client will try to login three times for each connection attempt. In other words, invoke the **ssh** client from the command line twice in order to trigger the `DROP` rule. On your third attempt, you will get a `connection refused` message.

```
$ ssh student@main.example.com
student@main.example.com's password:
Permission denied, please try again.
student@main.example.com's password:
Permission denied, please try again.
student@main.example.com's password:
Permission denied (publickey,gssapi-keyex,
  gssapi-with-mic,password).
$ ssh student@main.example.com
student@main.example.com's password:
Permission denied, please try again.
student@main.example.com's password:
Permission denied, please try again.
student@main.example.com's password:
Permission denied (publickey,gssapi-keyex,
  gssapi-with-mic,password).
$ ssh student@main.example.com
sh: connect to host main.example.com port 22:
  Connection refused
```

Chapter 16

Response and Mitigation

16.1 Labs

Lab 16

Exercise 16.1 Booting From Trusted Media

The ability to boot from trusted media is critical when attempting to analyze a system that has been compromised: We do not want to accidentally destroy or taint any evidence.

For this exercise, we are going to use the **Kali Linux** distribution, which is a fork of the **Backtrack Linux** distribution. This is a live **Linux** distribution that is designed from the ground to be a forensic tool for security professionals.

We will use the `main.example.com` for this exercise and make use of the **aide** information created in an earlier exercise to see what has changed on our suspected compromised system.

Review the documentation for the virtualization tool you are using for instructions on attaching the ISO to the VM. In general, to boot a live distribution in a virtualized environment, the VM must be configured to see our `*.iso` image first, in lieu of the normally configured root device. Depending on your environment, the steps necessary to achieve this may vary, but in most cases (by selecting the `settings` of your virtual machine) you will have the option to configure storage devices, as well as select a disk **image** (ISO file) as a physical disk, which in this case will allow you to boot the VM from the live **Kali** ISO.

Contained in the `Linux Foundation` **cmLFS216** directory is a copy of the **Kali** live iso:

`https://training.linuxfoundation.org/cm/LFS416/kali-linux-light-2016.1-amd64.iso`

The live iso may also be downloaded from:

`http://cdimage.kali.org/kali-2016.1/kali-linux-light-2016.1-amd64.iso`

Once the system is booted from the live iso, you will want to open a terminal and inspect the system to determine what physical devices are present. This can be achieved by executing the following from a privileged console.

```
# fdisk -l
```

This command will list the storage devices present on the system. This will allow you to determine how many disk(s) and partition(s) are present on the system. In our case we are not concerned with the actual partitions, only the physical devices (disks) themselves.

If there is only a single disk, it will be listed as `/dev/sda` with partitions listed as numbers following the device name, e.g., `/dev/sda1`, etc.

To make an image of the device in question, in this case, it will most likely be `/dev/sdb` (`/dev/sda` is the root partition of our live distribution), perform the following:

Note: This is an example only, do not run this command.

```
# dd if=/dev/sdb of=disk_image.iso bs=4096
```

This command will use **dd** to write the entire contents of our compromised system disk to an image file, which can then be mounted read-only.

To mount the image we created, use **mount** with the read-only and loopback options (`ro,loop`). For example:

Note: This is an example only, do not run this command.

```
# mount -o ro,loop disk_image.img /mnt
```

Once the image is mounted, additional copies can be made, and forensic analysis can be performed without fear of tainting the original evidence.

Another option we could use, is **chroot** on the **real** disk. We would want to mount the suspect disk read only. This assumes there is an **aide** database on the disk in question (a previous exercise).

Mount the root filesystem read only.

```
# mount -o ro /dev/sda1 /mnt
```

Use the **chroot** command to switch our root (\) to the disk in question.

```
# chroot /mnt
```

Now we can run the `aide --check` command to verify the disks contents without the possibility of altering the disk. The `aide --check` command will complain that it cannot record results in its log file, as expected.

```
# aide --check
```