CONTENTS

LAB 1 - Installing a virtual machine in Virtualbox & initial configuration

Lab Objectives:

In this lab, you will learn how to set up a virtual machine using VirtualBox, install various Linux distributions (Ubuntu Server 22.04, CentOS 7, AlmaLinux 8, and AlmaLinux 9), configure essential initial settings, and prepare your virtual machines for subsequent exercises. The lab objectives include:

- Installing VirtualBox and the VirtualBox Extension Pack.
- Downloading Linux distribution ISO files for Ubuntu Server 22.04, CentOS 7, AlmaLinux 8, and AlmaLinux 9.
- Creating a virtual machine with proper settings, such as allocating 20 GB of virtual disk space.
- Installing Ubuntu Server 22.04, CentOS 7, and AlmaLinux on separate virtual machines.
- Updating the Ubuntu virtual machine using apt.
- Configuring SSH on the Ubuntu virtual machine.
- Installing the EPEL repository on the CentOS 7 virtual machine.
- Configuring network settings to use Bridged Adapter mode for virtual machines.
- Creating a snapshot of your virtual machine for easy recovery.
- Demonstrating how to use the Windows SSH client to connect to Linux virtual machines.

1.1 – Installing virtual machine

Step 1: Download and install VirtualBox and the VirtualBox Extension Pack. You can get them from https://www.virtualbox.org/.

Step 2: Download the installation .iso files for Ubuntu Server 22.04, CentOS 7, AlmaLinux8, and AlmaLinux9. You can get them from https://ubuntu.com/, https://almalinux.org/, and https://www.centos.org/.

Step 3: Start VirtualBox and click the New icon at the top of the screen. Fill out the information where requested. Increase the virtual drive size to 20 GB, but leave everything else as the default settings, as shown here:

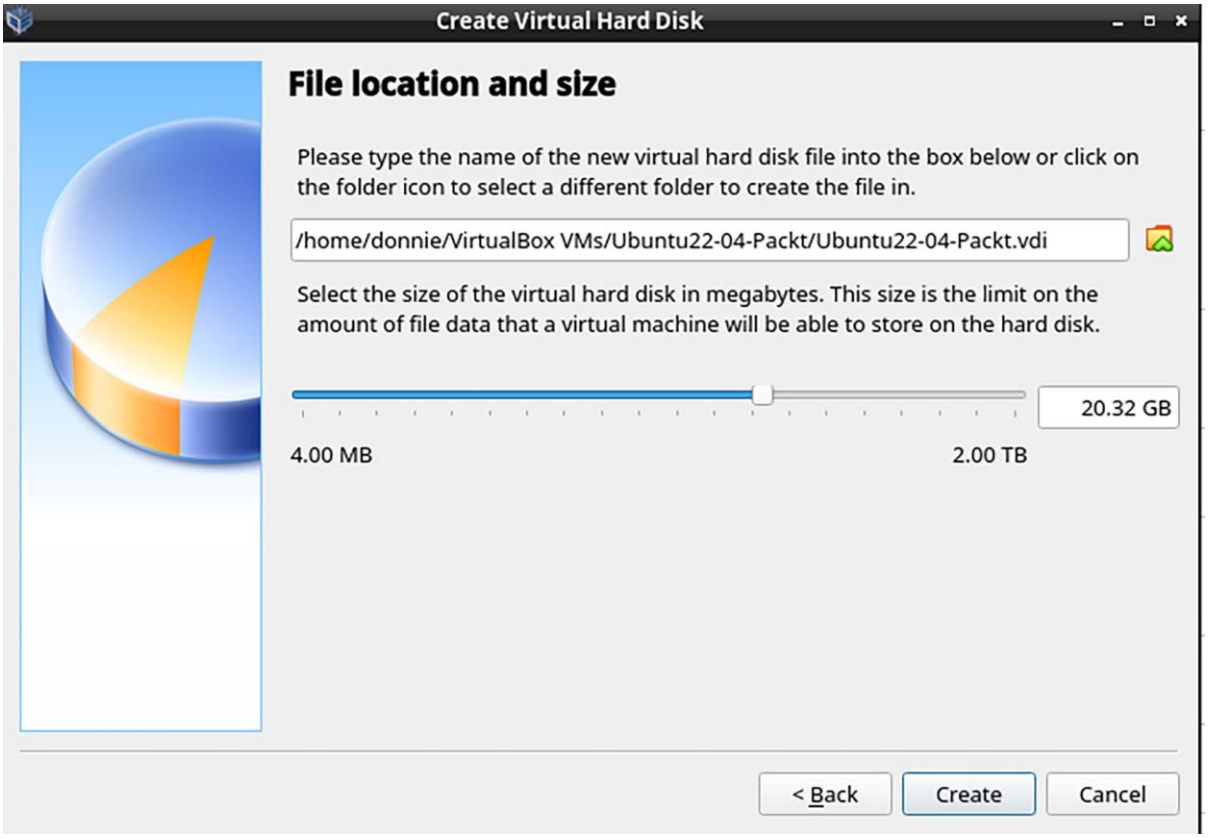


Figure 1.1: Create the virtual drive

Step 4: Start the new virtual machine. Click on the folder icon that's beside the Location dialog box and navigate to the directory where you stored the .isofiles that you downloaded. Choose either the Ubuntu ISO file, the CentOS ISO file, or one of the AlmaLinux ISO files, as shown in the following screenshot. (If the ISO file doesn't show up in the list, click the Add button in the top-lef corner to add it.)

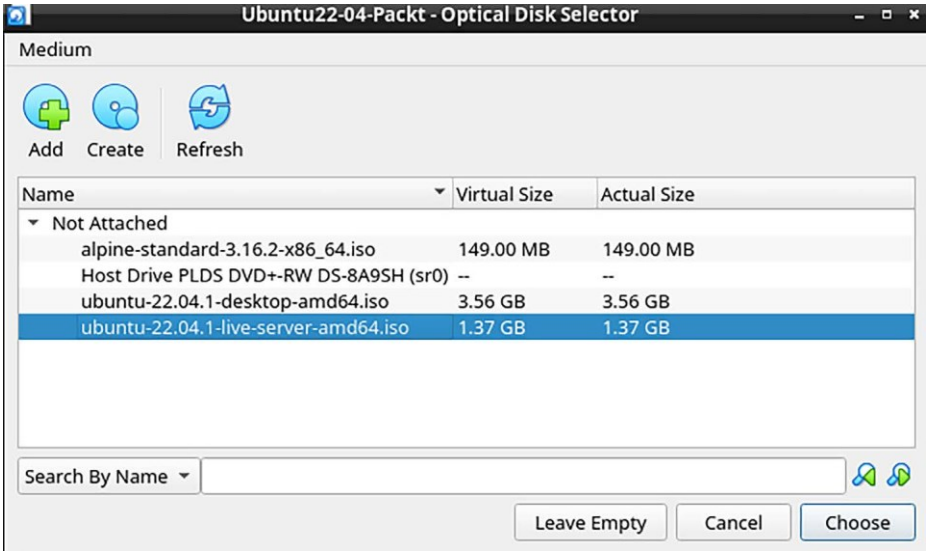


Figure 1.2: Choose the .iso file

Step 5: Click the Start button on the dialog box to start installing the operating system. Note that for Ubuntu Server, you won't be installing a desktop interface. For the CentOS 7 and AlmaLinux virtual machines, choose to install a Server without a graphical interface. (Later on, we'll go through at least one exercise that will require a desktop interface for an AlmaLinux machine. You can create a virtual machine with a graphical interface at that time.)

Step 6: When installing Ubuntu, choose Try or Install Ubuntu Server when you get to this screen:

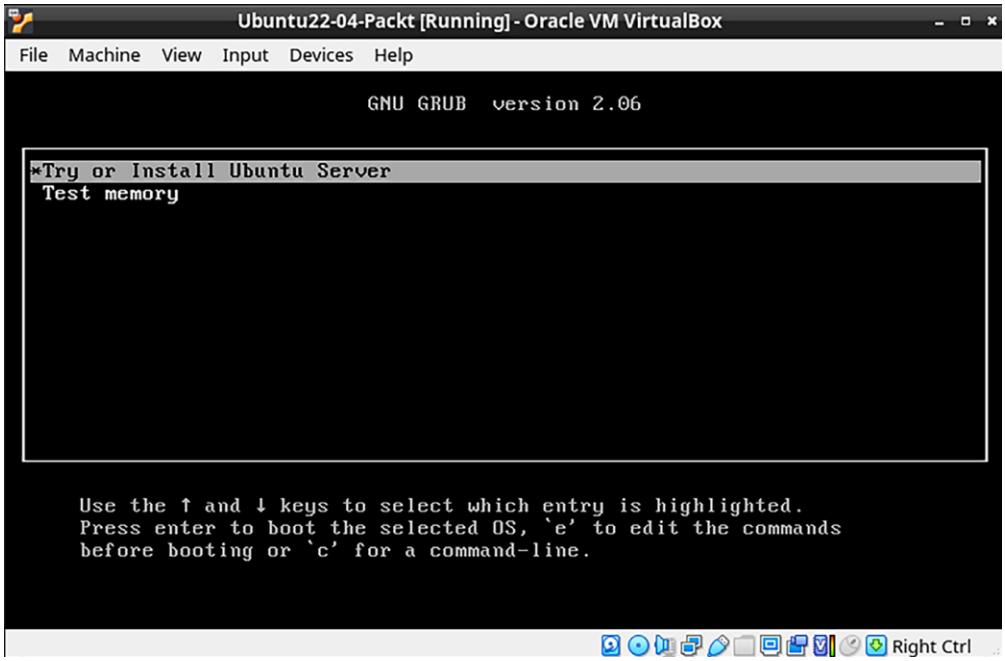


Figure 1.3: Installing Ubuntu

Step 7: Repeat the procedure for the other Linux distros.

Step 8: Update the Ubuntu virtual machine with these two commands:

```
sudo apt update
sudo apt dist-upgrade
```

Step 9: Hold off on updating the CentOS and AlmaLinux virtual machines because we'll do that in the next exercise.

Step 10: For Ubuntu, choose to install the OpenSSH Server on the SSH setup screen.

When installing Ubuntu, you'll be asked to create a normal user account and password for yourself. It won't ask you to create a root user password, but will instead automatically add you to the sudo group so that you'll have admin privileges.

When you get to the user account creation screen of the CentOS or AlmaLinux installer, be sure to check the Make this user administrator box for your own user account, since it isn't checked by default. It will offer you the chance to create a password for the root user, but that's entirely optional.

The user account creation screen of the AlmaLinux 9 installer—which looks the same as the one on CentOS 7 and AlmaLinux 8—is shown here:

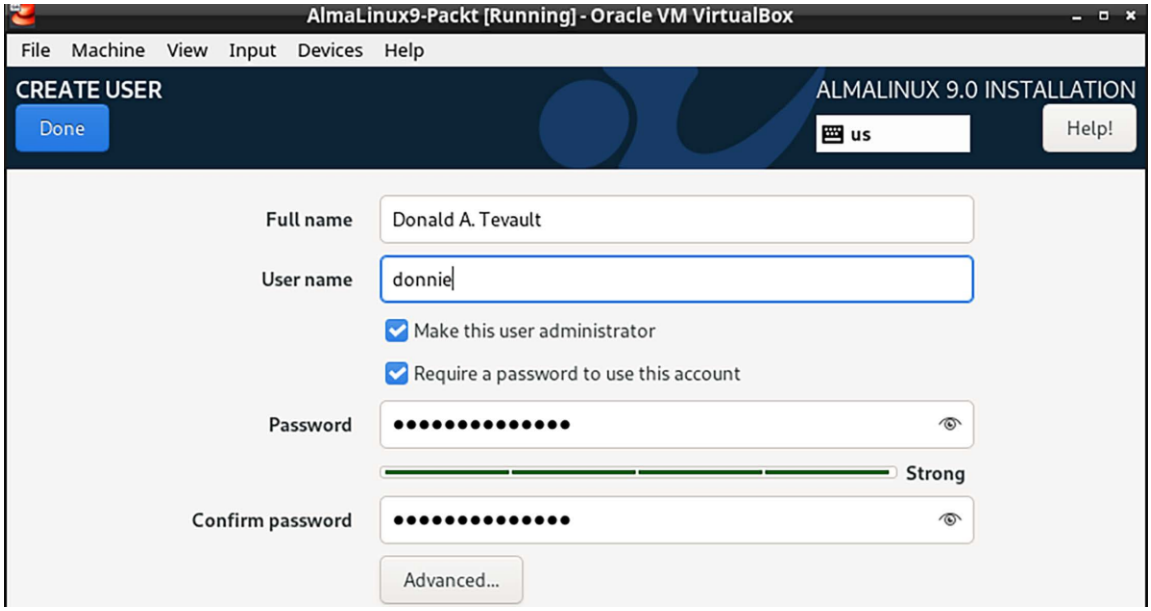


Figure 1.4: User creation for AlmaLinux

For Ubuntu 22.04, you'll see just one self-explanatory screen to set up your real name, a username, and a password. The Ubuntu installer will automatically add your user account to the sudo group, which will give you full administrator privileges. Here's the user account creation screen for Ubuntu 22.04:

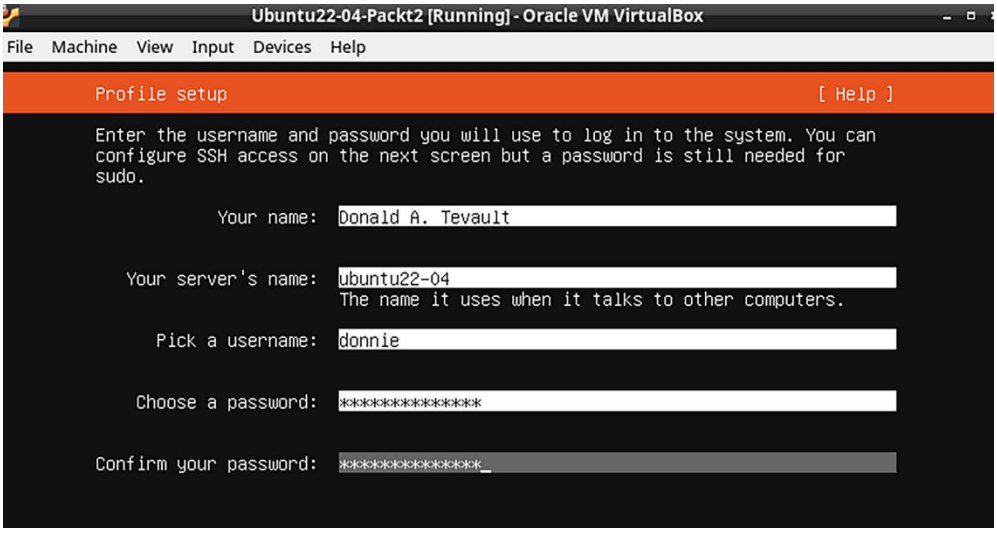


Figure 1.5: Ubuntu user creation

1.2 – Installing the EPEL repository on the Centos 7 virtual machine

While the Ubuntu package repositories have pretty much everything that you need for this course, the CentOS and AlmaLinux package repositories are—shall we say—lacking. To have the packages that you'll need for the CentOS and AlmaLinux hands-on labs, you'll need to install the EPEL repository. (The EPEL project is run by the Fedora team.) When you install third-party repositories on Red Hat 7 and CentOS 7 systems, you'll also need to install a priorities package and edit the .repo files to set the proper priorities for each repository. This will prevent packages from the third-party repository from overwriting official Red Hat and CentOS packages if they just happen to have the same name. The following steps will help you install the required packages and edit the .repo files:

Step 1: The two packages that you'll need to install EPEL are in the normal CentOS 7 repositories. To install them, just run this command:

```
sudo yum install yum-plugin-priorities epel-release
```

Step 2: When the installation completes, navigate to the /etc/yum.repos.d directory, and open the CentOS-Base.repo file in your favorite text editor. After the last line of the base , updates , and extras sections, add the line priority=1 . After the last line of the centosplus section, add the line priority=2 . Save the file and close the editor. Each of the sections that you've edited should look something like this, except with the appropriate name and priority number:

```
[base]
name=CentOS-$releasever – Base
mirrorlist=http://mirrorlist.centos.org/?
release=$releasever&arch=$basearch&repo=os&infra=$infra
#baseurl=http://mirror.centos.org/centos/
$releasever/os/$basearch/
gpgcheck=1
gpgkey=file:///etc/pki/rpm-gpg/RPM-GPG-KEY-CentOS-7
priority=1
```

Step 3: Open the epel.repo file for editing. After the last line of the epel section, add the line priority=10 . After the last line of each remaining section, add the line priority=11.

Step 4: Update the system and then create a list of the installed and available packages by running the following commands:

```
sudo yum upgrade
sudo yum list > yum_list.txt
```

1.3 – Configuring a network for Virtualbox virtual machines

Some of our training scenarios will require you to simulate creating a connection to a remote server. You would do this by using your host machine to connect to a virtual machine. When you first create a virtual machine on VirtualBox, the networking is set to NAT mode. In order to connect to the virtual machine from the host, you'll need to set the virtual machine's network adapter to Bridged Adapter mode. Here's how you can do this:

Step 1: Shut down any virtual machines that you've already created.

Step 2: On the VirtualBox Manager screen, open the Settings dialog for a virtual machine.

Step 3: Click the Network menu item. Change the Attached to setting from NAT to Bridged Adapter, and change the Promiscuous Mode setting to Allow All, as shown in this screenshot:

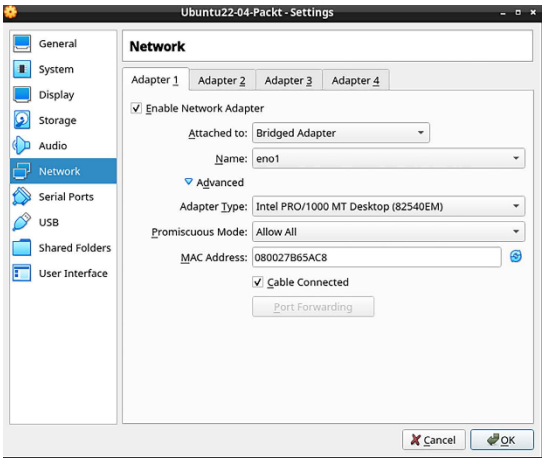


Figure 1.6: Configuring the network

Step 4: Restart the virtual machine and set it to use a static IP address.

Tip

If you assign static IP addresses from the high end of your subnet range, it will be easier to prevent conflicts with low-number IP addresses that get handed out from your internet gateway.

1.4 – Creating a virtual machine snapshot with Virtualbox

One of the beautiful things about working with virtual machines is that you can create a snapshot and roll back to it if you mess something up. With VirtualBox, that's easy to do, by following these steps:

Step 1: From the Machine menu of the VirtualBox Manager screen, select Tools/Snapshots.

Step 2: Further right on the screen, click on the Take icon to bring up the Snapshot dialog box. Either fill in the desired Snapshot Name or accept the default name. Optionally, you can create a description, as you see in this screenshot:

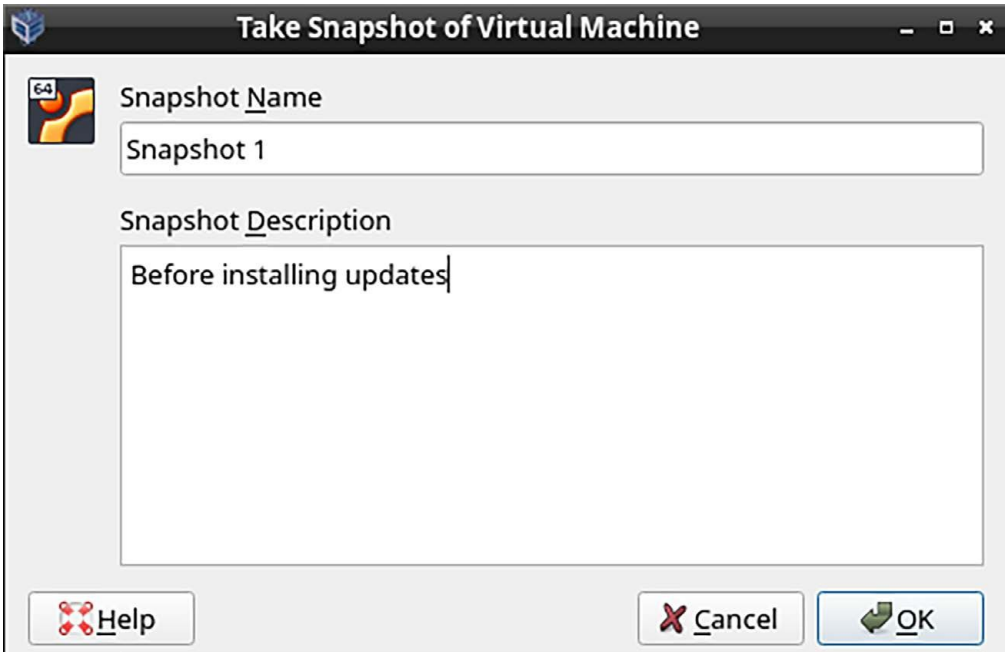


Figure 1.7: Taking a snapshot

After you've made changes to the virtual machine, you can roll back to the snapshot by shutting down the virtual machine, then highlighting the snapshot name, and clicking on the Restore button.

1.5 – Using the Windows SSH client to interface with Linux virtual machines

If you're using Windows, you already have an SSH client built into your operating system. So, let's see how to do this:

Step 1: To get to it, you can open the Terminal from the Windows System menu, like so:

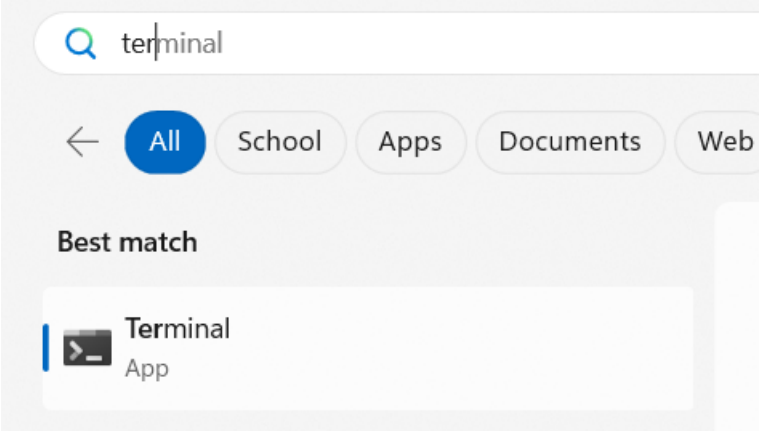
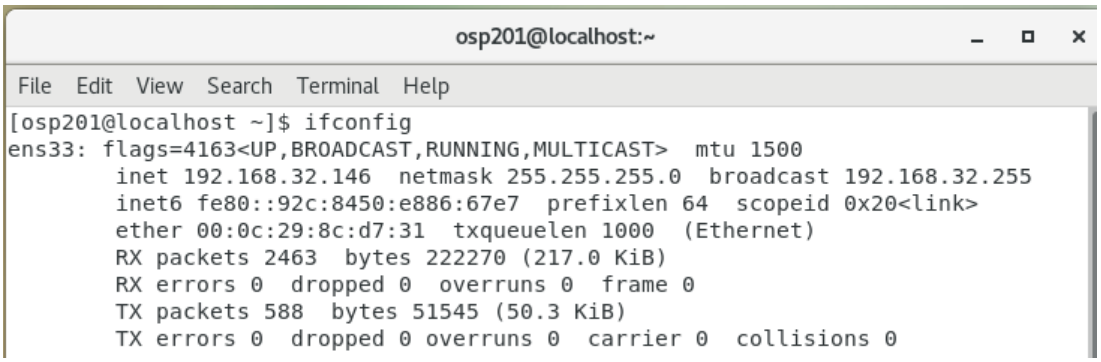


Figure 1.11: Windows 11 Terminal

Step 2: Open the Terminal in CentOS7 and type **ifconfig** to check the ip of the CentOS virtual machine.



Step 3: Then, just type in the SSH commands the same as you would from a Mac or Linux machine, like this:

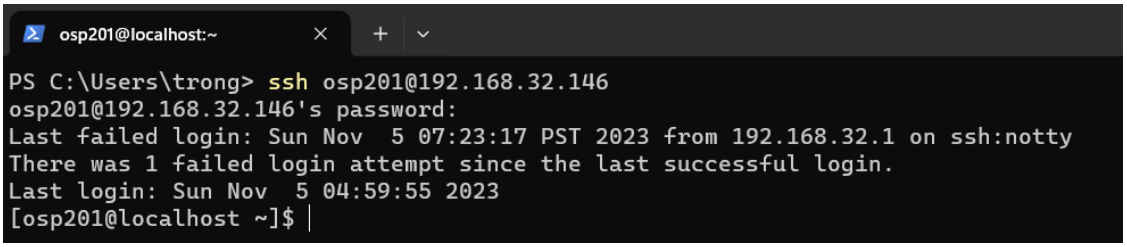


Figure 1.12: SSH remote from Windows Terminal

Step 4: A better option is to use Windows PowerShell instead of the normal Command Prompt. Get to it as you see here:

Figure 1.13: PowerShell command prompt

Step 4: As before, let's use it to log in to my Orange Pi device, as you see here:

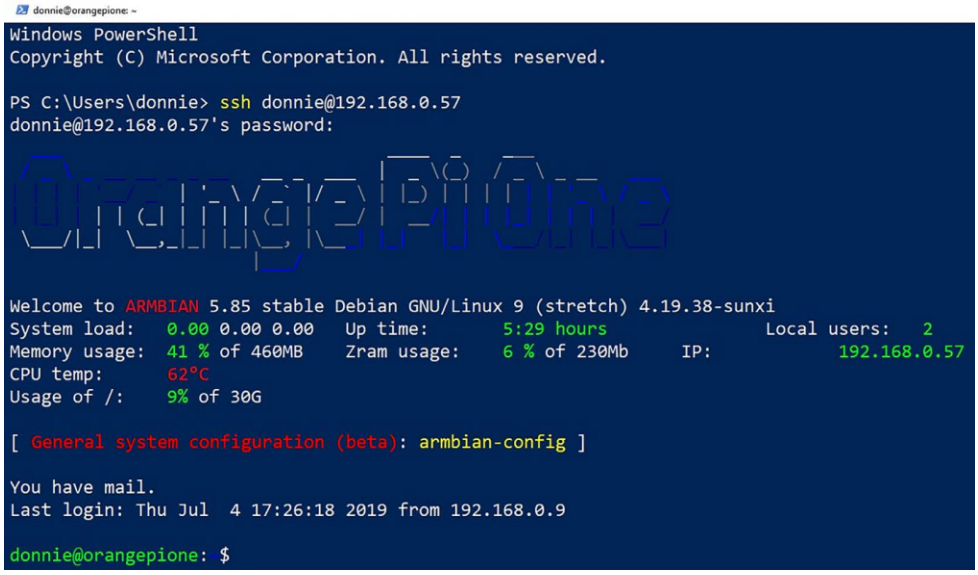


Figure 1.14: Remote login from PowerShell

If you have the choice, go with PowerShell instead of Command Prompt. PowerShell is a bit closer to the Linux Bash shell experience, and you'll be much happier with it.

Lab Summary:

In this lab, you have gained hands-on experience with virtualization and Linux distribution installation within VirtualBox. You have learned to configure the initial settings of virtual machines, update packages, and configure essential network settings. Additionally, you practiced taking snapshots for system recovery and interfacing with Linux virtual machines from a Windows environment using SSH.

By completing this lab, you are now equipped with fundamental skills required for future exercises and a solid foundation for working with virtual machines and Linux operating systems. These skills are invaluable for various IT and cybersecurity tasks, making you well-prepared for real-world scenarios.

LAB 2 - Configuring Basic Security Controls on a CentOS Linux Server

Lab Objectives:

In this lab series, you will configure basic security controls on a CentOS Linux server using various firewall tools, including iptables, nftables, and firewalld. The objectives for each part of the lab are as follows:

4.1 – Basic iptables usage:

- Understand the basics of iptables.
- Disable the Uncomplicated Firewall (ufw) and work directly with iptables.
- Create a basic firewall configuration allowing SSH access, DNS queries, and ICMP, while denying all other incoming traffic.
- Verify the iptables rules and make them persistent.
- Practice using snapshots for system recovery.

4.2 – Blocking invalid IPv4 packets:

- View existing rules in filter and mangle tables.
- Block invalid IPv4 packets and TCP packets that don't have the SYN flag set.
- Observe the effects of blocking invalid packets.
- Improve firewall efficiency using the PREROUTING chain of the mangle table.

4.3 – Configure ip6tables:

- Create an IPv6 firewall alongside an existing IPv4 firewall.
- Block invalid IPv6 packets and TCP packets without the SYN flag.
- Test the IPv6 firewall with Nmap scans.
- Observe the rules and packet counters for IPv6.

4.4 – Configure nftables on Ubuntu:

- Set up nftables on an Ubuntu virtual machine.
- Create a custom firewall configuration with nftables.
- Test the firewall by performing Nmap scans.
- Observe the rules and packet counters in the nftables configuration.
- Learn to block invalid packets with nftables.

4.5 – Basic ufw usage:

- Work with the Uncomplicated Firewall (ufw) on Ubuntu.
- Enable, configure, and disable ufw.
- Open specific ports and services using ufw.

- Modify ufw configuration files for advanced rules.
- Observe the effects of ufw rules on iptables or nftables.

4.6 – Firewalld commands:

- Learn firewalld basics on CentOS or AlmaLinux.
- Explore firewalld zones, services, and predefined rules.
- Modify the default zone and add services.
- Block ICMP types, log denied packets, and set firewall rules.
- View, compare, and make runtime and permanent firewall configurations.
- Use direct rules to block invalid packets and observe their effects.
- Test firewall configurations with Nmap scans.
- Explore firewalld-related man pages.

4.1 – Basic iptables usage

You'll complete this lab on your Ubuntu 20.04 virtual machine. Follow these steps to get started:

1. Shut down your Ubuntu virtual machine and create a snapshot. After you boot it back up, look at your iptables rules, or lack thereof, by using this command:

```
sudo iptables -L
```

2. With a default Ubuntu setup, the Uncomplicated Firewall (ufw) service is already running, albeit with an unactivated firewall configuration. We want to disable it to work directly with iptables. Do that now with this command:

```
sudo systemctl disable --now ufw
```

3. Create the rules that you need for a basic firewall, allowing for Secure Shell access, DNS queries and zone transfers, and the proper types of ICMP. Deny everything else:

```
sudo iptables -A INPUT -m conntrack --ctstate ESTABLISHED,RELATED -j
ACCEPT
sudo iptables -A INPUT -p tcp --dport ssh -j ACCEPT
sudo iptables -A INPUT -p tcp --dport 53 -j ACCEPT
sudo iptables -A INPUT -p udp --dport 53 -j ACCEPT
sudo iptables -A INPUT -m conntrack -p icmp --icmp-type 3 --ctstate
NEW,ESTABLISHED,RELATED -j ACCEPT
sudo iptables -A INPUT -m conntrack -p icmp --icmp-type 11 --ctstate
```

NEW,ESTABLISHED,RELATED -j ACCEPT

sudo iptables -A INPUT -m conntrack -p icmp --icmp-type 12 --ctstate

NEW,ESTABLISHED,RELATED -j ACCEPT

sudo iptables -A INPUT -j DROP

4. Use this command to view the results:

sudo iptables -L

5. Oops – it looks like you forgot about that loopback interface. Add a rule for it at the top of the list:

sudo iptables -I INPUT 1 -i lo -j ACCEPT

6. View the results by using the following two commands. Note the difference between the output of each:

sudo iptables -L

sudo iptables -L -v

7. Install the iptables-persistent package and choose to save the IPv4 and IPv6 rules when prompted:

sudo apt install iptables-persistent

8. Reboot the virtual machine and verify that your rules are still active.

9. Keep this virtual machine; you'll be adding more to it in the next hands-on lab.

That's the end of this lab—congratulations!

4.2 – Blocking invalid IPv4 packets

For this lab, you'll use the same virtual machine that you used for the previous lab. You won't replace any of the rules that you already have. Rather, you'll just add a couple. Let's get started:

1. Look at the rules for the filter and the mangle tables. (Note that the -v option shows you statistics about packets that were blocked by the DROP and REJECT rules.) Then, zero out the blocked packets counter:

sudo iptables -L -v

sudo iptables -t mangle -L -v

sudo iptables -Z

sudo iptables -t mangle -Z

2. From either your host machine or another virtual machine, perform the NULL and Windows Nmap scans against the virtual machine:

```
sudo nmap -sN ip_address_of_your_VM
sudo nmap -sW ip_address_of_your_VM
```

3. Repeat Step 1. You should see a large jump in the number of packets that were blocked by the final DROP rule in the INPUT chain of the filter table:

```
sudo iptables -L -v
sudo iptables -t mangle -L -v
```

4. Make the firewall work more efficiently by using the PREROUTING chain of the mangle table to drop invalid packets, such as those that are produced by the two Nmap scans that we just performed. Add the two required rules with the following two commands:

```
sudo iptables -t mangle -A PREROUTING -m conntrack --ctstate INVALID -j
DROP
sudo iptables -t mangle -A PREROUTING -p tcp ! --syn -m conntrack
--ctstate NEW -j DROP
```

5. Save the new configuration to your own home directory. Then, copy the file to its proper location and zero out the blocked packet counters:

```
sudo iptables-save > rules.v4
sudo cp rules.v4 /etc/iptables
sudo iptables -Z
sudo iptables -t mangle -Z
```

6. Perform only the NULL scan against the virtual machine:

```
sudo nmap -sN ip_address_of_your_VM
```

7. Look at the iptables ruleset and observe which rule was triggered by the Nmap scan:

```
sudo iptables -L -v
sudo iptables -t mangle -L -v
```

8. This time, perform just the Windows scan against the virtual machine:

```
sudo nmap -sW ip_address_of_your_VM
```

9. Observe which rule was triggered by this scan:

```
sudo iptables -L -v
sudo iptables -t mangle -L -v
```

That's the end of this lab—congratulations!

4.3 – Configure ip6tables

For this lab, you'll use the same Ubuntu virtual machine that you used in the previous iptables labs. You'll leave the IPv4 firewall setup that's already there as-is and create a new firewall for IPv6. Let's get started:

1. View your IPv6 rules, or lack thereof, with this command:

```
sudo ip6tables -L
```

2. Create the IPv6 firewall. Due to formatting constraints, I can't list the entire code block of commands here. You can find the respective commands in this chapter's directory, in the code file that you can download from the Packt Publishing website.

3. View the new ruleset by using the following command:

```
sudo ip6tables -L
```

4. Next, set up the mangle table rules for blocking invalid packets:

```
sudo ip6tables -t mangle -A PREROUTING -m conntrack --ctstate INVALID -j
DROP
sudo ip6tables -t mangle -A PREROUTING -p tcp ! --syn -m conntrack
--ctstate NEW -j DROP
```

5. Save the new ruleset to a file in your own home directory, and then transfer the rules file to the proper location:

```
sudo ip6tables-save > rules.v6
sudo cp rules.v6 /etc/iptables/
```

6. Obtain the IPv6 address of the virtual machine with this command:

```
ip a
```

7. On the machine on which you installed Nmap, perform a Windows scan of the virtual machine's IPv6 address. The command will look like this, except with your own IP address:

```
sudo nmap -6 -sW fe80::a00:27ff:fe9f:d923
```

8. On the virtual machine, observe which rule was triggered with this command:

```
sudo ip6tables -t mangle -L -v
```

You should see non-zero numbers for the packet counters for one of the rules.

1. On the machine on which you installed Nmap, perform an XMAS scan of the virtual machine's IPv6 address. The command will look like this, except with your own IP address:

```
sudo nmap -6 -sX fe80::a00:27ff:fe9f:d923
```

2. As before, on the virtual machine, observe which rule was triggered by this scan:

```
sudo ip6tables -t mangle -L -v
```

3. Shut down the virtual machine, and restore it from the snapshot that you created at the beginning of the Hands-on lab for basic iptables usage section.

That's the end of this lab – congratulations!

4.4 – Configure nftables on Ubuntu

For this lab, you'll need a clean snapshot of your Ubuntu 22.04 virtual machine. Let's get started. Restore your Ubuntu virtual machine to a clean snapshot to clear out any firewall configurations that you created previously. (Or, if you prefer, start with a new virtual machine.) Disable ufw and verify that no firewall rules are present:

```
sudo systemctl disable —now ufw
sudo iptables -L
```

You should see no rules listed for nftables.

Copy the workstation.nft template over to the /etc/ directory and rename it nftables.conf :

```
sudo cp /usr/share/doc/nftables/examples/syntax/workstation /etc/nftables.conf
```

Edit the /etc/nftables.conf file to create your new configuration. (Note that due to formatting constraints, I have to break this into three different code blocks.) Make the top portion of the file look like this:

```
#!/usr/sbin/nft -f flush ruleset
table inet filter {
chain prerouting {
type filter hook prerouting priority 0;
ct state invalid counter log prefix "Invalid Packets: " drop
tcp flags & (fin|syn|rst|ack) != syn ct state new counter log prefix
"Invalid Packets 2: " drop
```

```
}
```

Make the second portion of the file look like this:

```
chain input {
type filter hook input priority 0;
# accept any localhost traffic
iif lo accept
# accept traffic originated from us
ct state established,related accept
# activate the following line to accept common local services
tcp dport 22 ip saddr { 192.168.0.7, 192.168.0.10 } log prefix "Blocked
SSH packets: " drop
tcp dport { 22, 53 } ct state new accept
udp dport 53 ct state new accept
ct state new,related,established icmp type { destination-unreachable,
time-exceeded, parameter-problem } accept
```

Make the final portion of the file look like this:

```
ct state new,related,established icmpv6 type { destination-unreachable,
time-exceeded, parameter-problem } accept
# accept neighbour discovery otherwise Ipv6 connectivity breaks.
ip6 nexthdr icmpv6 icmpv6 type { nd-neighbor-solicit, nd-router-
advert, nd-neighbor-advert } accept
# count and drop any other traffic
counter log prefix "Dropped packet: " drop
}
}
```

Save the file and reload nftables :

```
sudo systemctl reload nftables
```

View the results:

```
sudo nft list tables
sudo nft list tables
```

sudo nft list table inet filter

sudo nft list ruleset

From either your host computer or from another virtual machine, do a Windows scan against the Ubuntu virtual machine:

sudo nmap -sW ip_address_of_UbuntuVM

Look at the packet counters to see which blocking rule was triggered. (Hint: It's in the prerouting chain.):

sudo nft list ruleset

This time, do a null scan of the virtual machine:

sudo nmap -sN ip_address_of_UbuntuVM

Finally, look at which rule was triggered this time. (Hint: It's the other one in the prerouting chain.):

sudo nft list ruleset

In the /var/log/kern.log file, search for the Invalid Packets text string to view the messages about the dropped invalid packets.

That's the end of this lab – congratulations!

4.5 – Basic ufw usage

You'll need to complete this lab on a clean snapshot of either an Ubuntu 20.04 or an Ubuntu 22.04 virtual machine. Let's get started:

1. Shut down your Ubuntu virtual machine and restore the snapshot to get rid of all of the iptables or nftables stuff that you just did. (Or, if you prefer, just start with a fresh virtual machine.)

2. When you've restarted the virtual machine, verify that the iptables rules are now gone. On Ubuntu 20.04 do:

sudo iptables -L

On Ubuntu 22.04, do:

sudo nft list ruleset

3. View the status of ufw. Open port 22/TCP and then enable ufw. Then, view the results:

sudo ufw status

sudo ufw allow 22/tcp

sudo ufw enable

sudo ufw status

On Ubuntu 20.04, do:

sudo iptables -L

sudo ip6tables -L

On Ubuntu 22.04, do:

sudo nft list ruleset

4. This time, open port 53 for both TCP and UDP:

sudo ufw allow 53

sudo ufw status

On Ubuntu 20.04, do:

sudo iptables -L

sudo ip6tables -L

On Ubuntu 22.04, do:

sudo nft list ruleset

5. cd into the /etc/ufw/ directory. Familiarize yourself with the contents of the files that are there.

Open the /etc/ufw/before.rules file in your favorite text editor. At

the bottom of the file, below the COMMIT line, add the following code

snippet:

```
# Mangle table added by Donnie
*mangle
:PREROUTING ACCEPT [0:0]
-A PREROUTING -m conntrack --ctstate INVALID -j DROP
-A PREROUTING -p tcp -m tcp ! --tcp-flags FIN,SYN,RST,ACK SYN -m
conntrack --ctstate NEW -j DROP
COMMIT
```

(Note that the second PREROUTING command wraps around on the printed page.)

6. Repeat step 6 for the /etc/ufw/before6.rules file.

7. Reload the firewall with this command:

```
sudo ufw reload
```

8. On Ubuntu 20.04, observe the rules by doing:

```
sudo iptables -L
sudo iptables -t mangle -L
sudo ip6tables -L
sudo ip6tables -t mangle -L
```

On Ubuntu 22.04, observe the rules by doing:

```
sudo nft list ruleset
```

9. Take a quick look at the ufw status:

```
sudo ufw status
```

That's the end of the lab – congratulations!

4.6 – Firewalld commands

By completing this lab, you'll get some practice with basic firewalld commands:

1. Log into your CentOS 7 virtual machine or either of the AlmaLinux virtual machines and run the following commands. Observe the output after each one:

```
sudo firewall-cmd --get-zones
sudo firewall-cmd --get-default-zone
sudo firewall-cmd --get-active-zones
```

2. Briefly view the man pages that deal with firewalld.zones:

```
man firewalld.zones
man firewalld.zone
```

(Yes, there are two of them. One explains the zone configuration files, while the other explains the zones themselves.)

3. Look at the configuration information for all of the available zones:

```
sudo firewall-cmd --list-all-zones
```

4. Look at the list of predefined services. Then, look at the information about the dropbox-lansync service:

```
sudo firewall-cmd --get-services
sudo firewall-cmd --info-service=dropbox-lansync
```

5. Set the default zone to dmz. Look at the information concerning the zone, add the http and https services, and then look at the zone information again:

```
sudo firewall-cmd --set-default-zone=dmz
sudo firewall-cmd --permanent --add-service={http,https}
sudo firewall-cmd --info-zone=dmz
sudo firewall-cmd --permanent --info-zone=dmz
```

6. Reload the firewall configuration and look at the zone information again. Also, look at the list of services that are being allowed:

```
sudo firewall-cmd --reload
sudo firewall-cmd --info-zone=dmz
sudo firewall-cmd --list-services
```

7. Permanently open port 10000/tcp and view the results:

```
sudo firewall-cmd --permanent --add-port=10000/tcp
sudo firewall-cmd --list-ports
sudo firewall-cmd --reload
sudo firewall-cmd --list-ports
sudo firewall-cmd --info-zone=dmz
```

8. Remove the port that you just added:

```
sudo firewall-cmd --permanent --remove-port=10000/tcp
sudo firewall-cmd --reload
sudo firewall-cmd --list-ports
sudo firewall-cmd --info-zone=dmz
```

9. Add a rich language rule to block a geographic range of IPv4 addresses:

```
sudo firewall-cmd --permanent --add-rich-rule='rule family="ipv4" source
address="200.192.0.0/24" service name="http" drop'
```

10. Block the host-redirect and network-redirect ICMP types:

```
sudo firewall-cmd --permanent --add-icmp-block={host-redirect,network-redirect}
```

11. Add the directive to log all dropped packets:

```
sudo firewall-cmd --set-log-denied=all
```

12. View both the runtime and permanent configurations and note the differences between them:

```
sudo firewall-cmd --info-zone=dmz
sudo firewall-cmd --info-zone=dmz --permanent
```

13. Make the runtime configuration permanent and verify that it took effect:

```
sudo firewall-cmd --runtime-to-permanent
sudo firewall-cmd --info-zone=dmz --permanent
```

14. On CentOS 7, view the complete list of effective firewall rules by doing:

```
sudo iptables -L
```

15. On AlmaLinux 8 or 9, view the complete list of effective firewall rules by doing:

```
sudo nft list ruleset
```

16. Create the direct rules in order to block invalid packets from the mangle table's PREROUTING chain:

```
sudo firewall-cmd --direct --add-rule ipv4 mangle PREROUTING 0 -m
conntrack --ctstate INVALID -j DROP
sudo firewall-cmd --direct --add-rule ipv4 mangle PREROUTING 1 -p tcp !
--syn -m conntrack --ctstate NEW -j DROP
sudo firewall-cmd --direct --add-rule ipv6 mangle PREROUTING 0 -m
conntrack --ctstate INVALID -j DROP
sudo firewall-cmd --direct --add-rule ipv6 mangle PREROUTING 1 -p tcp !
--syn -m conntrack --ctstate NEW -j DROP
```

17. Verify that the rules took effect and make them permanent:

```
sudo firewall-cmd --direct --get-rules ipv4 mangle PREROUTING
sudo firewall-cmd --direct --get-rules ipv6 mangle PREROUTING
sudo firewall-cmd --runtime-to-permanent
```

18. View the contents of the direct.xml file that you've just created:

```
sudo less /etc/firewalld/direct.xml
```

19. Perform XMAS Nmap scans for both IPv4 and IPv6 against the virtual machine. Then, observe which rule was triggered by the scan:

sudo nmap -sX ipv4_address_of_Test-VM

sudo nmap -6 -sX ipv6_address_of_Test-VM

sudo iptables -t mangle -L -v

sudo ip6tables -t mangle -L -v

20. Repeat step 19, but this time with a Windows scan:

sudo nmap -sW ipv4_address_of_Test-VM

sudo nmap -6 -sW ipv6_address_of_Test-VM

sudo iptables -t mangle -L -v

sudo ip6tables -t mangle -L -v

21. View the list of main pages for firewalld:

apropos firewall

That's the end of the lab – congratulations!

Lab Summary:

In this comprehensive lab series, you have gained hands-on experience with various firewall tools and configurations on Linux servers. You have learned how to set up and manage firewalls using iptables, nftables, and firewalld. Key objectives include configuring rules, blocking invalid packets, enabling services, and monitoring firewall activities.

These practical exercises have equipped you with essential skills for securing Linux systems and managing network access. Understanding and configuring firewalls are crucial aspects of maintaining security in any IT environment.

LAB 3 - Securing administrative and normal user accounts

Lab Objectives:

In this lab, you will learn and practice essential security techniques for user account management and password policies on a Linux system. The objectives of this lab include:

1. **Assigning Limited Sudo Privileges:**
    - Create user accounts for specific users.
    - Configure sudo privileges for each user with varying levels of access.
    - Verify the effectiveness of the assigned sudo privileges.
2. **Disabling the Sudo Timer:**
    - Disable the sudo timeout feature to require password input for every sudo command.
    - Understand the impact of different timeout configurations.
3. **Setting Password Complexity Criteria:**
    - Configure password complexity criteria, including length and character classes.
    - Test password assignments against the defined criteria.
4. **Setting Account and Password Expiry Data:**
    - Create user accounts with specified expiration dates.
    - Modify account expiration dates.
    - Configure account aging parameters such as password change waiting periods and expiration.
5. **Preventing Brute-Force Password Attacks by Configuring pam_faillock on Ubuntu:**
    - Configure PAM modules to prevent brute-force password attacks.
    - Understand the faillock configuration and its impact on security.
6. **Detecting Compromised Passwords:**
    - Utilize the pwnedpasswords API to check the security status of passwords.
    - Create a script to automate password security checks.

2.1 – Assigning limited sudo privileges

In this lab, you'll create some users and assign them different levels of privileges. To simplify things, let's use the CentOS7 virtual machine:

Step 1: Log in to the CentOS7 virtual machine and create user accounts for Lionel, Katelyn, and Maggie:

```
sudo useradd lionel
sudo useradd katelyn
```

```
sudo useradd Maggie
sudo passwd lionel
sudo passwd katelyn
sudo passwd Maggie
```

Step 2: Open visudo :

```
sudo visudo
```

Find the STORAGE command alias and remove the comment symbol from in front of it.

Step 3: Add the following lines to the end of the file, using tabs to separate the columns:

```
lionel ALL=(ALL) ALL
katelyn ALL=(ALL) /usr/bin/systemctl status sshd
maggie ALL=(ALL) STORAGE
```

Save the file and exit visudo .

Step 4: To save time, we'll use su to log in to the different user accounts. That way, you won't need to log out of your own account to perform these steps. First, log in to Lionel's account and verify that he has full sudo privileges by running several root-level commands:

```
su – lionel
sudo su –
exit
sudo systemctl status sshd
sudo fdisk -l
exit
```

Step 5: This time, log in as Katelyn and try to run some root-level commands. Don't be too disappointed if they don't all work, though:

```
su – katelyn
sudo su –
sudo systemctl status sshd
sudo systemctl restart sshd
sudo fdisk -l
exit
```

Step 6: Finally, log in as Maggie, and run the same set of commands that you ran for Katelyn.

Step 7: Keep in mind that, although we only had three individual users for this lab, you could just as easily have handled more users by setting them up in user aliases or Linux groups.

2.2 – Disabling the sudo timer

For this lab, you'll disable the sudo timer on your AlmaLinux VM:

Step 1: Log in to the same AlmaLinux virtual machine that you used for the previous lab. We'll be using the user accounts that you've already created.

Step 2: At your own user account command prompt, enter the following commands:

```
sudo fdisk -l
sudo systemctl status sshd
sudo iptables -L
```

You'll see that you only needed to enter the password once to do all three commands

Step 3: At your own user account command prompt, run the following:

```
sudo fdisk -l
sudo -k
sudo fdisk -l
```

Note how the sudo -k command resets your timer, so you'll have to enter your password again.

Open visudo with the following command:

```
sudo visudo
```

In the Defaults specification section of the file, add the following line:

```
Defaults timestamp_timeout = 0
```

Save the file and exit visudo .

Step 4: Perform the commands that you performed in step 2. This time, you should see that you have to enter a password every time.

Step 5: Open visudo and modify the line that you added so that it looks like this:

```
Defaults:lionel timestamp_timeout = 0
```

Save the file and exit visudo .

Step 6: From your own account shell, repeat the commands that you performed in step 2. Then, log in as Lionel and perform the commands again.

Step 7: View your own sudo privileges by running the following:

```
sudo -l
```

3.2 – Setting password complexity criteria

For this lab, you can use either a CentOS, AlmaLinux, or Ubuntu virtual machine, as desired. The only difference is that you won't perform Step 1 for either CentOS or AlmaLinux:

Step 1: For Ubuntu only, install the libpam-pwquality package:

```
sudo apt install libpam-pwquality
```

Step 2: Open the /etc/security/pwquality.conf file in your preferred text editor. Remove the comment symbol from in front of the minlen line and change the value to 19 . It should now look like this:

```
minlen = 19
```

Step 3: Save the file and exit the editor.

Step 4: Create a user account for Goldie and attempt to assign her the passwords turkeylips, TurkeyLips , and Turkey93Lips. Note the change in each warning message.

Step 5: In the pwquality.conf file, comment out the minlen line. Uncomment the minclass line and the maxclassrepeat line. Change the maxclassrepeat value to 5 . The lines should now look like this:

```
minclass = 3
maxclassrepeat = 5
```

Step 6: Save the file and exit the text editor.

Step 7: Try assigning various passwords that don't meet the complexity criteria that you've set to Goldie's account and view the results.

• In the /etc/login.defs file on your CentOS 7 machine, you'll see the line PASS_MIN_LEN 5.

• Supposedly, this is to set the minimum password length, but in reality, pwquality overrides it. So, you could set this value to anything at all, and it would have no effect. (Note that the PASS_MIN_LEN parameter is no longer supported on RHEL 8/9-type distros.)

3.3 – Setting account and password expiry data

Step 1: On your CentOS or AlmaLinux VM, create a user account for Samson with the expiration date of June 30, 2025, and view the results:

```
sudo useradd -e 2025-06-30 samson
sudo chage -l samson
```

Step 2: For Ubuntu, run these commands:

```
sudo useradd -m -d /home/samson -s /bin/bash -e 2025-06-30 samson
sudo chage -l samson
```

Step 3: Use usermod to change Samson's account expiration date to July 31, 2025:

```
sudo usermod -e 2025-07-31 samson
sudo chage -l samson
```

Step 4: Assign a password to Samson's account, then force him to change his password on his first login. Log in as Samson, change his password, then log in to your own account:

```
sudo passwd samson
sudo passwd -e samson
sudo chage -l samson
su - samson
exit
```

Step 5: Use chage to set a five-day waiting period for changing passwords, a password expiration period of 90 days, an inactivity period of two days, and a warning period of five days:

```
sudo chage -m 5 -M 90 -I 2 -W 5 samson
sudo chage -l samson
```

Step 6: Keep this account because you'll be using it for the lab in the next section.

3.5 – Preventing brute-force password attacks by configuring pam_faillock on Ubuntu

1. Open the /etc/pam.d/common-auth file in your favorite text editor. At the top of the file, insert these two lines:

auth required pam_faillock.so preauth silent

auth required pam_faillock.so authfail

2. Open the /etc/pam.d/common-account file in your text editor. At the bottom of the file, add this line:

account required pam_faillock.so

3. Configure the /etc/security/faillock.conf file the same way that I showed you in Step 5 of the preceding lab for AlmaLinux.

4. Test the setup as outlined in Steps 6 through 8 of the preceding AlmaLinux lab.

5. And that's all there is to it. Next, let's look at how to manually lock a user's account.

3.6 – Detecting compromised passwords

In this lab, you'll use the pwnedpasswords API in order to check your own passwords:

1. Use curl to see how many passwords there are with the 21BD1 string in their password hashes:

```
curl https://api.pwnedpasswords.com/range/21BD1
```

2. In the home directory of any of your Linux virtual machines, create the pwnpassword.sh script with the following content:

```
#!/bin/bash
candidate_password=$1
echo "Candidate password: $candidate_password"
full_hash=$(echo -n $candidate_password | sha1sum | awk '{print
substr($1, 0, 32)}')
prefix=$(echo $full_hash | awk '{print substr($1, 0, 5)}')
suffix=$(echo $full_hash | awk '{print substr($1, 6, 26)}')
if curl https://api.pwnedpasswords.com/range/$prefix | grep -i $suffix;
then echo "Candidate password is compromised";
else echo "Candidate password is OK for use";
fi
```

3. Add the executable permission to the script:

```
chmod u+x pwnedpasswords.sh
```

4. Run the script, specifying TurkeyLips as a password:

```
./pwnedpasswords.sh TurkeyLips
```

5. Repeat Step 4 as many times as you like, using a different password each time.

The user management techniques that we've looked at so far work great on a small number of computers. But what if you're working in a large enterprise? We'll look at that next.

Lab Summary:

In this lab, you have gained hands-on experience in securing user accounts and enhancing password policies on Linux systems. Key takeaways from this lab include:

- Assigning selective sudo privileges to users for controlled access.
- Managing the sudo timeout to enhance system security.
- Configuring password complexity criteria to enforce strong passwords.
- Managing account and password expiration dates for user accounts.
- Implementing security measures to prevent brute-force password attacks.
- Utilizing external services like the pwnedpasswords API to detect compromised passwords.

These skills are crucial for ensuring the security and integrity of user accounts and access control in Linux environments.

LAB 4 – Applying Hardened Linux File System Security Controls

Lab Objectives:

In this lab, you will explore various aspects of securing Linux file systems, SSH configurations, and system logging. The objectives of this lab include:

6.1 – Creating and Transferring SSH Keys:

- Generate SSH key pairs and secure them with a passphrase.
- Transfer the public SSH key to a remote server for passwordless authentication.
- Explore the usage of SSH key pairs for secure login.

6.2 – Disabling Root Login and Password Authentication:

- Modify SSH server configurations to disable root login and password authentication.
- Enhance SSH security by limiting login methods.

6.3 – Setting Up Two-Factor Authentication on Ubuntu 22.04:

- Install and configure Google Authenticator for two-factor authentication.
- Apply two-factor authentication to both local terminal login and sudo operations.

6.4 – Using Google Authenticator with Key Exchange on Ubuntu:

- Integrate Google Authenticator with SSH key-based authentication.
- Require both key-based authentication and Google Authenticator verification for enhanced security.

6.6 – Disabling Weak SSH Encryption Algorithms – Ubuntu 22.04:

- Identify and disable weak SSH encryption algorithms to improve security.
- Verify the impact of algorithm changes on SSH connectivity.

6.9 – Configuring More Verbose SSH Logging:

- Adjust SSH log levels for enhanced verbosity.
- Analyze SSH logs to gain insights into SSH connection details.

6.10 – Configuring Whitelists Within sshd_config:

- Create user accounts and groups on a Linux VM.
- Configure SSH whitelists using AllowUsers and AllowGroups directives.
- Test SSH access restrictions based on user accounts and groups.

7.1 – Searching for SUID and SGID Files:

- Use the **find** command to identify and list files with SUID and SGID permissions.
- Understand the security implications of SUID and SGID files.

7.2 – Setting Security-Related Extended File Attributes:

- Create a file and set extended file attributes using **chattr**.
- Explore and test the effects of immutable (**i**) and append-only (**a**) attributes.
- Understand how extended attributes enhance file security.

6.1 – Creating and transferring SSH keys

In this lab, you'll use one virtual machine (VM) as your client, and one VM as the server. Alternatively, if you're using a Windows host machine, you can use Cygwin, PowerShell, or the built-in Windows Bash shell for the client. (Be aware, though, that PowerShell and the Windows Bash shell store the key files in alternate locations.) If you're on either a Mac or a Linux host machine, you can use the host machine's native command-line terminal as the client. In any case, the procedure will be the same.

For the server VM, use either Ubuntu 22.04 or CentOS 7.

Let's get started:

1. On the client machine, create a pair of 384-bit elliptic curve keys. Accept the default filename and location and create a passphrase:

```
ssh-keygen -t ecdsa -b 384
```

2. Observe the keys, taking note of the permissions settings:

```
ls -l ./ssh
```

3. Add your private key to your session keyring. Enter your passphrase when prompted:

```
exec /usr/bin/ssh-agent $SHELL
ssh-add
```

4. Transfer the public key to the server VM. When prompted, enter the password for your user account on the server VM. (Substitute your own username and IP address in the following command.):

```
ssh-copy-id donnie@192.168.0.7
```

5. Log in to the server VM as you normally would:

```
ssh donnie@192.168.0.7
```

6. Observe the authorized_keys file that was created on the server VM:

```
ls -l .ssh
cat .ssh/authorized_keys
```

7. Log out of the server VM and close the terminal window on the client. Open another terminal window and try to log in to the server again. This time, you should be prompted to enter the passphrase for your private key.
8. Log back out of the server VM and add your private key back to the session keyring of your client. Enter the passphrase for your private key when prompted:

```
exec /usr/bin/ssh-agent $SHELL
ssh-add
```

9. As long as you keep this terminal window open on your client, you'll be able to log in to the server VM as many times as you want without having to enter a password. However, when you close the terminal window, your private key will be removed from your session keyring.
10. Keep your server VM, because we'll do more with it in a bit.
You've reached the end of the lab – congratulations!

6.2 – Disabling root login and password authentication

For this lab, use the same server VM that you used for the previous lab. Let's get started:
1. On either an Ubuntu, CentOS, or AlmaLinux 8 server VM, look for this line in the sshd_config file:

```
#PasswordAuthentication yes
```

2. Remove the comment symbol, change the parameter value to no , and reload the SSH daemon. The line should now look like this:

```
PasswordAuthentication no
```

3. Now, when the botnets scan your system, they'll see that doing a brute-force password attack would be useless. They'll then just go away and leave you alone.
4. Look for either of these two lines, depending on whether the server is an Ubuntu or a CentOS 7/AlmaLinux VM:

```
#PermitRootLogin yes
#PermitRootLogin prohibit-password
```

Uncomment the line and change it to the following:

```
PermitRootLogin no
```

5. Reload the SSH daemon so that it will read in the new changes. On Ubuntu, do this:

```
sudo systemctl reload ssh
```

6. On CentOS/AlmaLinux, do this:

```
sudo systemctl reload sshd
```

7. Attempt to log in to the server VM from the client that you used in the previous lab.

8. Attempt to log in to the server VM from another client on which you haven't created a key pair. (You shouldn't be able to.)

9. As before, keep the server VM, because we'll do more with it in a bit.

You've reached the end of the lab – congratulations!

6.3 – Setting up two-factor authentication on Ubuntu 22.04

For this lab, start with a fresh Ubuntu 22.04 VM that's not set up for public key authentication. (That will save a lot of confusion when going through this procedure):

1. Install Google Authenticator on your smart phone. (It's in the normal app stores for both Android and iPhone.)

2. On your Ubuntu VM, install the libpam-google-authenticator package, like this:

```
sudo apt install libpam-google-authenticator
```

3. For this step, if you haven't already, use SSH to remotely log in to the Ubuntu VM from the GUI-type terminal of your host machine. (That's because you might need to resize things to make the next step work.) Now, from this GUI-type terminal, run the google-authenticator app, like so:

```
google-authenticator
```

A big QR code will now show up on your screen. If the whole code graphic isn't visible, use your GUI terminal controls to zoom out until the whole graphic is visible.

Bring up the Google Authenticator app on your smart phone, and touch the + sign in the lower right-hand corner of the screen. Choose the Scan a QR code option, and then take a picture of your QR code.

On your smart phone, note that a new entry for your Ubuntu VM has been added to the list. On the Ubuntu VM, enter the verification code that shows up with that entry.

The next thing you'll see on the Ubuntu terminal is your emergency scratch codes. Copy them down and store them in a safe location. (If you lose your mobile phone, you'll use these scratch codes to log in.)

4. Next, you'll be asked a series of questions. Just enter y for everything.

In this step, you'll set up two-factor authentication for logging in at the local terminal and for using sudo . Open the /etc/pam.d/common-auth file in your favorite text editor. Add the auth required pam_google_authenticator.so line as the first parameter. The top portion of the file should now look something like this:

```
# /etc/pam.d/common-auth - authentication settings common to all services
#
# This file is included from other service-specific PAM config files,
. . .
. . .
auth required pam_google_authenticator.so
# here are the per-package modules (the "Primary" block)
. . .
. . .
```

At the local terminal of the Ubuntu VM, log out and then log back in. When prompted, enter the verification code from your smart phone app. Perform a command that requires sudo privileges. You should see something like this:

```
donnie@ubuntu2204-packt:~$ sudo nft list ruleset
Verification code:
[sudo] password for donnie:
. . .
. . .
Enter the verification code at the prompt.
```

From either your host machine or another VM, remotely log in to the Ubuntu VM via SSH. You should still be able to do that because we haven't yet configured the /etc/ssh/sshd_config file. Open the sshd_config file in your text editor, and change the KbdInteractiveAuthentication no line to KbdInteractiveAuthentication yes .

5. Reload the Secure Shell configuration:

```
sudo systemctl reload ssh
```

Try logging in again from either your host machine or another VM. This time, you should be prompted to enter your verification code.

6. Now, let's say that your organization needs two-factor authentication for remote SSH logins, but doesn't need it for either local logins or sudo operations. Let's change the configuration so that only remote users will need to enter a code. Open the /etc/pam.d/common-auth file in your text editor, and remove the line that you inserted in step 9.

7. Open the /etc/pam.d/sshd file in your text editor, and add that line just under the @include common-auth line at the top of the file. The top portion of the file should now look like this:

```
PAM configuration for the Secure Shell service
# Standard Un*x authentication.
@include common-auth
auth required pam_google_authenticator.so
```

8. You should now be able to log in to the local terminal and perform sudo actions without having to enter a verification code. Instead, you should only have to enter a verification code when logging in remotely.
9. End of lab.

6.4 – Using Google Authenticator with key exchange on Ubuntu

For this lab, use the same Ubuntu VM that you used for the previous lab:
1. On either your host machine or another VM, create a pair of keys and transfer them to the Ubuntu VM, as you did in the Creating and transferring SSH keys lab. This time, you should be prompted to enter a verification code when you execute the ssh-copy-id command.
2. On the Ubuntu VM, open the /etc/ssh/sshd_config file in your text editor. This time, instead of changing the #PasswordAuthentication yes line, add this line below the KbdInteractiveAuthentication yes line:

```
AuthenticationMethods publickey keyboard-interactive:pam
```

3. After reloading the SSH configuration, you'll see that you'll be able to remotely log in by using key exchange if you're logging in from a machine that has that set up. If you're logging in from a machine that doesn't have key exchange set up, you'll still be able to log in with a password and a verification code. So, we don't have true two-factor authentication just yet.
4. To require both key-based authentication and Google Authenticator verification, change the above line to look like this:

```
AuthenticationMethods publickey,keyboard-interactive:pam
```

5. After reloading the SSH configuration, you'll only be allowed to log in from machines for which you've set up key exchange. You now effectively have three-factor authentication because you'll still be prompted to enter your normal login password.

6. To disable the password login so that you'll only be using key exchange and a verification code, open the /etc/pam.d/sshd file in your text editor. At the very top of the file, find the @include common-auth line and change it to #@include common-auth.

7. Verify that the key exchange works by trying to log in from a VM on which you haven't performed the key exchange setup. (You shouldn't be allowed to.)

8. That's it. End of lab.

6.6 – Disabling weak SSH encryption algorithms – Ubuntu 22.04

For this lab, you'll need the VM that you've been using as a scanner, and another Ubuntu 22.04 VM to scan and configure. Let's get started:

1. If you haven't done so already, scan the Ubuntu 22.04 VM and save the output to a file:

```
nmap --script=ssh2-enum-algos.nse 192.168.0.14 -oN ubuntuscan.txt
```

2. Count the number of lines in the file by doing:

```
wc -l ubuntuscan.txt
```

3. On the target Ubuntu 22.04 VM, open the /etc/ssh/sshd_config file in your preferred text editor. Toward the top of the file, find these two lines:

```
# Ciphers and keying
#RekeyLimit default none
```

4. Beneath those two lines, insert these three lines:

```
Ciphers -aes128-ctr,aes192-ctr,aes128-gcm@openssh.com
KexAlgorithms ecdh-sha2-nistp384
MACs -hmac-sha1-etm@openssh.com,hmac-sha1,umac-64-etm@openssh.
com,umac-64@openssh.com,umac-128-etm@openssh.com,umac-128@openssh.
com,hmac-sha2-256-etm@openssh.com,hmac-sha2-256
```

5. In the Ciphers and MACs lines, you see a comma-separated list of algorithms that were disabled by the preceding - sign. (You only need one - to disable all the algorithms in the list.) In the KexAlgorithms line, there's no - sign. This means that the algorithm that's listed on that line is the only one that is enabled.

6. Save the file and restart the SSH daemon. Verify that it started correctly:

sudo systemctl restart ssh

sudo systemctl status ssh

7. Scan the Ubuntu 22.04 VM again, saving the output to a different file:

nmap --script=ssh2-enum-algos.nse 192.168.0.14 -oN ubuntuscan_modified.txt

8. Count the number of lines in the new file:

wc -l ubuntuscan_modified.txt

9. On the scanner VM, use diff to compare the two files. You should see fewer algorithms than you saw previously:

diff -y ubuntuscan.txt ubuntuscan_modified.txt

6.9 – Configuring more verbose SSH logging

For this lab, use the same VM that you've been using for the previous labs. That way, you'll get a better picture of what a complete sshd_config file should look like when it's fully locked down. Remotely log in to the target VM via SSH and follow these steps:

1. Open the main log file and scroll down to where you see the entry that was made due to your login and observe what it says. For Ubuntu, do:

sudo less /var/log/auth.log

2. For CentOS or AlmaLinux, do:

sudo less /var/log/secure

3. As I mentioned previously, you never want to run a production machine with the SSH log level set to any of the DEBUG levels. But, just so you can see what it does log, set your machine to DEBUG now. Open the /etc/ssh/sshd_config file in your favorite text editor. Find the line that says the following:

#LogLevel INFO

4. Change it to the following:

LogLevel DEBUG3

5. After saving the file, reload SSH. On Ubuntu, do:

```
sudo systemctl reload ssh
```

6. On CentOS or AlmaLinux, do this:

```
sudo systemctl reload sshd
```

7. Log out of the SSH session, and then log back in. View the system log file to see the new entries from this new login.

8. Open the /etc/ssh/sshd_config file for editing. Change the LogLevel DEBUG3 line to the following:

```
LogLevel VERBOSE
```

9. After saving the file, reload or restart the SSH daemon. Log out of the SSH session, log back in, and look at the entries in the system log file.

6.10 – Configuring whitelists within sshd_config

This lab will work on any of your VMs. Follow these steps:

1. On the VM that you wish to configure, create user accounts for Frank, Charlie, and Maggie. On Ubuntu, do it like this:

```
sudo adduser frank
```

2. On CentOS or AlmaLinux, do it like this:

```
sudo useradd frank
sudo passwd frank
```

3. Create the webadmins group and add Frank to it:

```
sudo groupadd webadmins
sudo usermod -a -G webadmins frank
```

4. From either your host machine or from another VM, have the three users log in. Then, log them back out.

5. Open the /etc/ssh/sshd_config file in your favorite text editor. At the bottom of the file, add an AllowUsers line with your own username, like so:

```
AllowUsers donnie
```

6. Then, restart or reload the SSH service and verify that it has started correctly:

For Ubuntu:

```
sudo systemctl restart ssh
sudo systemctl status ssh
```

For CentOS:

```
sudo systemctl restart sshd
sudo systemctl status sshd
```

7. Repeat step 3. This time, these three kitties shouldn't be able to log in. Open the /etc/ssh/sshd_config file in your text editor. This time, add an AllowGroups line to the bottom of the file for the webadmins group, like so:

```
AllowGroups webadmins
```

8. Restart the SSH service and verify that it started properly.
9. From either your host machine or another VM, have Frank try to log in. You'll see that even though he's a member of the webadmins group, he'll still be denied. That's because the AllowUsers line with your own username takes precedence.
10. Open sshd_config in your text editor and remove the AllowUsers line that you inserted in step 4. Restart the SSH service and verify that it started properly.
11. Try to log in to your own account, and then try to log in to the accounts of all the other users. You should now see that Frank is the only one who is allowed to log in. The only way that any of the other users can now log in to the VM is from the VM's local console.
12. Log in to your own account at the VM's local console. Delete the AllowGroups line from sshd_config and restart the SSH service.
You've reached the end of the lab – congratulations!

7.1 – Searching for SUID and SGID files

You can perform this lab on either of your virtual machines. You'll save the output of the find command to a text file. Let's get started:
1. Search through the entire filesystem for all the files that have either SUID or SGID set before saving the output to a text file:

```
sudo find / -type f -perm /6000 -ls > suid_sgid_files.txt
```

2. Log into any other user account that you have on the system and create a dummy shell script file. Then, set the SUID permission on that file and log back out and back into your own user account:

```
su - desired_user_account
touch some_shell_script.sh
chmod 4755 some_shell_script.sh
ls -l some_shell_script.sh
exit
```

3. Run the find command again, saving the output to a different text file:

```
sudo find / -type f -perm /6000 -ls > suid_sgid_files_2.txt
```

4. View the difference between the two files:

```
diff suid_sgid_files.txt suid_sgid_files_2.txt
```

That's the end of the lab – congratulations!

7.2 – Setting security-related extended file attributes

For this lab, you'll need to create a perm_demo.txt file with some text of your choice. You'll set the i and a attributes and view the results. Let's get started:

1. Using your preferred text editor, create the perm_demo.txt file with a line of text.
2. View the extended attributes of the file:

```
lsattr perm_demo.txt
```

3. Add the a attribute:

```
sudo chattr +a perm_demo.txt
lsattr perm_demo.txt
```

4. Try to overwrite and delete the file:

```
echo "I want to overwrite this file." > perm_demo.txt
sudo echo "I want to overwrite this file." > perm_demo.txt
rm perm_demo.txt
sudo rm perm_demo.txt
```

5. Now, append something to the file:

```
echo "I want to append this line to the end of the file." >> perm_demo.txt
```

6. Remove the a attribute and add the i attribute:

```
sudo chattr -a perm_demo.txt
lsattr perm_demo.txt
sudo chattr +i perm_demo.txt
lsattr perm_demo.txt
```

7. Repeat Step 4.

8. Additionally, try to change the filename and create a hard link to the file:

```
mv perm_demo.txt some_file.txt
sudo mv perm_demo.txt some_file.txt
ln ~/perm_demo.txt ~/some_file.txt
sudo ln ~/perm_demo.txt ~/some_file.txt
```

9. Now, try to create a symbolic link to the file:

```
ln -s ~/perm_demo.txt ~/some_file.txt
```

That’s the end of the lab – congratulations!

Lab Summary:

In this lab series, you have gained hands-on experience in implementing various security controls and configurations to enhance the security of Linux systems and files. Key takeaways from this lab include:

- Establishing secure SSH key-based authentication and two-factor authentication.
- Disabling root login and strengthening password authentication.
- Configuring SSH encryption algorithms to mitigate vulnerabilities.
- Enhancing SSH logging for improved visibility and monitoring.
- Identifying and managing SUID and SGID files.
- Setting security-related extended file attributes for improved file system security.

These skills are essential for maintaining the integrity and security of Linux systems and file systems in diverse environments.

LAB 5 – Hardening Security for Linux Services and Applications

Lab Objectives:

1. **Understanding Linux Security Fundamentals**: Gain a comprehensive understanding of the fundamental concepts and principles of security in Linux systems.
2. **Implementing User Authentication**: Learn how to set up and manage user authentication mechanisms using tools like PAM (Pluggable Authentication Modules) and understand the importance of strong user authentication.
3. **Exploring File System Security**: Explore and implement file system security measures such as file permissions, access control lists (ACLs), and file attributes to enhance the security posture of a Linux system.
4. **Securing Network Communication**: Learn how to secure network communication by configuring firewalls, implementing secure shell (SSH) protocols, and applying best practices for secure data transfer.
5. **Deploying Encryption Techniques**: Understand the concepts of encryption and learn how to implement encryption techniques for data at rest and in transit to safeguard sensitive information.

5.1 – Creating your GPG keys

1. On a text-mode AlmaLinux machine, the first thing you need to do is to install the pinentry package. Do that with:

```
sudo dnf install pinentry
```

(Note that you won't have to do this with either a GUI-mode AlmaLinux machine or with Ubuntu Server.)

2. Next, create your pair of GPG keys:

```
gpg --full-generate-key
```

3. Note that since you're setting this up for yourself, you don't need sudo privileges.

The first thing that this command does is to create a populated .gnupg directory in your home directory:

```
gpg: /home/donnie/.gnupg/trustdb.gpg: trustdb created
gpg: key 56B59F39019107DF marked as ultimately trusted
gpg: directory '/home/donnie/.gnupg/openpgp-revocs.d' created
gpg: revocation certificate stored as
'/home/donnie/.gnupg/openpgprevocs.d/BD057E0E01E664424E8B812E56B59F39019107DF.rev'
public and secret key created and signed.
```

4. You'll then be asked to select which kinds of keys you want. We'll just go with the default RSA and RSA. RSA keys are stronger and harder to crack than the older DSA keys. Elgamal keys are good, but they may not be supported by older versions of GPG:

Please select what kind of key you want:
(1) RSA and RSA (default)
(2) DSA and Elgamal
(3) DSA (sign only)
(4) RSA (sign only)
(14) Existing key from card
Your selection?

5. For decent encryption, you'll want to go with a key of at least 3,072 bits, because anything smaller is now considered vulnerable. (This is according to the newest guidance from the U.S. National Institute of Standards and Technology, or NIST.) That's now the default on our newest Linux distros, so you're already good there. On older distros, such as CentOS 7, the default is only 2,048 bits, so you'll need to change it.

6. Next, select how long you want the keys to remain valid before they automatically expire. For our purposes, we'll go with the default key does not expire:

Please specify how long the key should be valid.
0 = key does not expire
<n> = key expires in n days
<n>w = key expires in n weeks
<n>m = key expires in n months
<n>y = key expires in n years
Key is valid for? (0)

7. Provide your personal information:

GnuPG needs to construct a user ID to identify your key.
Real name: Donald A. Tevault
Email address: donniet@something.net
Comment: No comment
You selected this USER-ID:
"Donald A. Tevault (No comment) <donniet@something.net>"
Change (N)ame, (C)omment, (E)mail or (O)kay/(Q)uit?

Create a passphrase for your private key:

You need a Passphrase to protect your secret key.

8. We need to generate a lot of random bytes. It is a good idea to perform some other action (type on the keyboard, move the mouse, utilize the disks) during the prime generation; this gives the random number generator a better chance to gain enough entropy.

9. On older Linux distros, this could take a while, even when you're doing all of the recommended things to create entropy. On newer Linux distros, the random number generator works more efficiently, so you can disregard the notice about how the key generation could take a long time. Here's what you'll see when the process has finished:

gpg: /home/donnie/.gnupg/trustdb.gpg: trustdb created

gpg: key 19CAEC5B marked as ultimately trusted

public and secret key created and signed.

gpg: checking the trustdb

gpg: 3 marginal(s) needed, 1 complete(s) needed, PGP trust model

gpg: depth: 0 valid: 1 signed: 0 trust: 0-, 0q, 0n, 0m, 0f, 1u

pub 2048R/19CAEC5B 2017-10-26

Key fingerprint = 8DE5 8894 2E37 08C4 5B26 9164 C77C 6944 19CA EC5B

uid Donald A. Tevault (No comment) <donniet@something.net>

sub 2048R/37582F29 2017-10-26

10. Verify that the keys did get created:

[donnie@localhost ~]$ gpg --list-keys

/home/donnie/.gnupg/pubring.gpg

---------- pub 2048R/19CAEC5B

2017-10-26

uid Donald A. Tevault (No comment) donniet@something.net

sub 2048R/37582F29 2017-10-26

[donnie@localhost ~]$

11. While you're at it, take a look at the files that you created:

[donnie@localhost ~]$ ls -l .gnupg/

total 12

drwx------. 2 donnie donnie 58 Oct 26 14:53 openpgp-revocs.d

```
drwx------. 2 donnie donnie  110 Oct 26 14:53 private-keys-v1.d
-rw-r--r--. 1 donnie donnie 1970 Oct 26 14:53 pubring.kbx
-rw-------. 1 donnie donnie   32 Oct 26 14:43 pubring.kbx~
-rw-------. 1 donnie donnie 1280 Oct 26 15:51 trustdb.gpg
[donnie@localhost ~]$
```

These files are your public and private keyrings, a revocation database, and a trusted users database.

5.2 – Symmetrically encrypting your own files

You may find GPG useful for encrypting your own files, even when you never plan to share them with anyone else. For this, you'll use symmetric encryption, which involves using your own private key for encryption. Before you try this, you'll need to generate your keys, as I outlined in the previous section:

Symmetric key encryption is, well, just that, symmetric. It's symmetric in the sense that the key that you would use to encrypt a file is the same key that you would use to decrypt the file. That's great for if you're just encrypting files for your own use. But if you need to share an encrypted file with someone else, you'll need to figure out a secure way to give that person the password. I mean, it's not like you'd want to just send the password in a plain-text email.

1. In addition to your own user account, you'll also need a user account for Maggie. On AlmaLinux, create her account like this:

```
sudo useradd maggie
sudo passwd maggie
```

For Ubuntu, create Maggie's account like this:

```
sudo adduser maggie
```

2. Let's encrypt a super-secret file that we just can't allow to fall into the wrong hands:

```
[donnie@localhost ~]$ gpg -c secret_squirrel_stuff.txt
[donnie@localhost ~]$
```

Note that the -c option indicates that I chose to use symmetric encryption with a passphrase for the file. The passphrase that you enter will be for the file, not for your private key.

3. Look at your new set of files. One slight flaw with this is that GPG makes an encrypted copy of the file, but it also leaves the original unencrypted file intact:

```
[donnie@localhost ~]$ ls -l
```

```
total 1748
-rw-rw-r--. 1 donnie donnie 37 Oct 26 14:22 secret_squirrel_stuff.txt
-rw-rw-r--. 1 donnie donnie 94 Oct 26 14:22
secret_squirrel_stuff.txt.gpg
[donnie@localhost ~]$
```

4. Let's get rid of that unencrypted file with shred. We'll use the -u option to delete the file, and the -z option to overwrite the deleted file with zeros:

```
[donnie@localhost ~]$ shred -u -z secret_squirrel_stuff.txt
[donnie@localhost ~]$
```

It doesn't look like anything happened, because shred doesn't give you any output. But ls -l will prove that the file is gone.

5. Now, if I were to look at the encrypted file with less secret_squirrel_stuff.txt.gpg, I would be able to see its contents after being asked to enter my private key passphrase. Try this for yourself:

```
less secret_squirrel_stuff.txt.gpg
Shhh!!!! This file is super-secret.
secret_squirrel_stuff.txt.gpg (END)
```

6. As long as my private key remains loaded into my keyring, I'll be able to view my encrypted file again without having to reenter the passphrase. Now, just to prove to you that the file really is encrypted, I'll create a shared directory, and move the file there for others to access. Again, go ahead and give it a try:

```
sudo mkdir /shared
sudo chown donnie: /shared
sudo chmod 755 /shared
mv secret_squirrel_stuff.txt.gpg /shared
```

When I go into that directory to view the file with less, I can still see its contents without having to reenter my passphrase.

7. But now, let's see what happens when Maggie tries to view the file. Use su - maggie to switch to her account, and have her try:

```
su - maggie
cd /shared
[maggie@localhost shared]$ less secret_squirrel_stuff.txt.gpg
```

"secret_squirrel_stuff.txt.gpg" may be a binary file. See it anyway?

And when she hits the Y key to see it anyway, she gets this:

<8C>^M^D^C^C^B<BD>2=<D3>u<93><CE><C9>MOOy<B6>^O<A2><AD>}
Rg9<94><EB><C4>^W^E

<A6><8D><B9><B8><D3>(<98><C4>æF^_8Q2b<B8>C<B5><DB>^]<F1><CD>#<90>H<EB><90><
C5>^S%X [<E9><EF><C7>
^@y+<FC><F2><BA><U+058C>H'+<D4>v<84>Y<98>G<D7>-
secret_squirrel_stuff.txt.gpg (END)

Poor Maggie really wants to see my file, but all she can see is encrypted gibberish.

What I've just demonstrated is another advantage of GPG. After entering your private key passphrase once, you can View any of your encrypted files without having to manually decrypt them, and without having to reenter your passphrase. With other symmetric file encryption tools, such as bcrypt, you wouldn't be able to view your files without manually decrypting them first.

8. But let's now say that you no longer need to have this file encrypted, and you want to decrypt it in order to let other people see it. Exit Maggie's account by typing exit. Then, just use gpg with the -d option:

[maggie@localhost shared]$ exit
[donnie@localhost shared]$ gpg -o secret_squirrel_stuff.txt -d secret_
squirrel_stuff.txt.gpg
gpg: AES256.CFB encrypted data
gpg: encrypted with 1 passphrase
Shhh!!!! This file is super-secret.
[donnie@localhost shared]$

This works differently from how it worked on older Linux distros. On our newer distros, we now have to use the -o option along with the filename of the decrypted file that we want to create. Also, note that the -o option has to come before the -d option, or else you'll get an error message.

5.3 – Encrypting files with public keys

In this lab, you'll learn about how to encrypt and share a file with GPG public key encryption:

1. To begin, create a user account for Frank, as you did for Maggie in the previous lab.

2. Create a key set for both yourself and for Frank, as I've already shown you. Next, extract your own public keys into an ASCII text file:

```
cd .gnupg
gpg --export -a -o donnie_public-key.txt
```

Log in as Frank, and repeat this command for him.

3. Normally, the participants in this would send their keys to each other either through an email attachment or by placing the keys in a shared directory. In this case, you and Frank will receive each other's public key files and place them into your respective .gnupg directories. Once that's done, import each other's keys:

```
donnie@ubuntu:~/.gnupg$ gpg --import frank_public-key.txt
gpg: key 4CFC6990: public key "Frank Siamese (I am a cat.) <frank@any.
net>" imported
gpg: Total number processed: 1
gpg: imported: 1 (RSA: 1)
donnie@ubuntu:~/.gnupg$
frank@ubuntu:~/.gnupg$ gpg --import donnie_public-key.txt
gpg: key 9FD7014B: public key "Donald A. Tevault <donniet@something.net>"
imported
gpg: Total number processed: 1
gpg:               imported: 1  (RSA: 1)
frank@ubuntu:~/.gnupg$
```

4. Now for the good stuff. Create a super-secret message for Frank, asymmetrically encrypt it (-e), and sign it (-s). Signing the message is the verification that the message really is from you, rather than from an impostor:

```
donnie@ubuntu:~$ gpg -s -e secret_stuff_for_frank.txt
. . .
. . .
It is NOT certain that the key belongs to the person named
in the user ID.  If you *really* know what you are doing,
you may answer the next question with yes.
Use this key anyway? (y/N) y
Current recipients:
2048R/CD8104F7 2017-10-27 "Frank Siamese (I am a cat.) <frank@any.net>"
```

```
Enter the user ID.  End with an empty line:
donnie@ubuntu:~$
```

So, the first thing you have to do is to enter the passphrase for your private key. Where it says to enter the user ID, enter frank, since he's the intended recipient of your message. But look at the line after that, where it says There is no assurance this key belongs to the named user. That's because you still haven't trusted Frank's public key. We'll get to that in a bit. The last line of the output again says to enter a user ID so that we can designate multiple recipients. But Frank is the only one you care about right now, so just hit the Enter key to break out of the routine. This results in a .gpg version of your message to Frank:

```
donnie@ubuntu:~$ ls -l
total 8
. . .
-rw-rw-r-- 1 donnie donnie 143 Oct 27 18:37 secret_stuff_for_frank.txt
-rw-rw-r-- 1 donnie donnie 790 Oct 27 18:39 secret_stuff_for_frank.txt.
gpg
donnie@ubuntu:~$
```

5. The final step on your end is to send Frank his encrypted message file by whatever means available.

6. When Frank receives his message, he'll use the -d option to view it:

```
frank@ubuntu:~$ gpg -d secret_stuff_for_frank.txt.gpg
. . .
. . .
gpg: gpg-agent is not available in this session
gpg: encrypted with 2048-bit RSA key, ID CD8104F7, created 2017-10-27
      "Frank Siamese (I am a cat.) <frank@any.net>"
This is TOP SECRET stuff that only Frank can see!!!!!
If anyone else see it, it's the end of the world as we know it.
(With apologies to REM.)
gpg: Signature made Fri 27 Oct 2017 06:39:15 PM EDT using RSA key ID
9FD7014B
gpg: Good signature from "Donald A. Tevault <donniet@something.net>"
gpg: WARNING: This key is not certified with a trusted signature!
gpg:          There is no indication that the signature belongs to the
```

```
owner.
Primary key fingerprint: DB0B 31B8 876D 9B2C 7F12  9FC3 886F 3357 9FD7
014B
frank@ubuntu:~$
```

Frank enters the passphrase for his private key, and he sees the message. At the bottom, he sees the warning about how your public key isn't trusted, and that There is no indication that the signature belongs to the owner. Let's say that you and Frank know each other personally, and he knows for a fact that the public key really is yours. He then adds your public key to the trusted list:

```
frank@ubuntu:~$ cd .gnupg
frank@ubuntu:~/.gnupg$ gpg --edit-key donnie
gpg (GnuPG) 1.4.20; Copyright (C) 2015 Free Software Foundation, Inc.
This is free software: you are free to change and redistribute it.
There is NO WARRANTY, to the extent permitted by law.
gpg: checking the trustdb
gpg: 3 marginal(s) needed, 1 complete(s) needed, PGP trust model
gpg: depth: 0  valid:   2  signed:   0  trust: 0-, 0q, 0n, 0m, 0f, 2u
pub  2048R/9FD7014B  created: 2017-10-27  expires: never      usage: SC
                     trust: ultimate      validity: ultimate
sub  2048R/9625E7E9  created: 2017-10-27  expires: never      usage: E
[ultimate] (1). Donald A. Tevault <donniet@something.net>
gpg>
```

8. The last line of this output is the command prompt for the gpg shell. Frank is concerned with trust, so he'll enter the trust command:

```
gpg> trust
pub  2048R/9FD7014B  created: 2017-10-27  expires: never      usage: SC
                     trust: unknown       validity: unknown
sub  2048R/9625E7E9  created: 2017-10-27  expires: never      usage: E
[ unknown] (1). Donald A. Tevault <donniet@something.net>
Please decide how far you trust this user to correctly verify other
users' keys
(by looking at passports, checking fingerprints from different sources,
```

etc.)

1 = I don't know or won't say

2 = I do NOT trust

3 = I trust marginally

4 = I trust fully

5 = I trust ultimately

m = back to the main menu

Your decision? 5

Do you really want to set this key to ultimate trust? (y/N) y

9. Frank has known you for quite a while, and he knows for a fact that you're the one who sent the key. So, he chooses option 5 for ultimate trust. Once Frank logs out and logs back in, that trust will take effect:

```
frank@ubuntu:~$ gpg -d secret_stuff_for_frank.txt.gpg
You need a passphrase to unlock the secret key for
user: "Frank Siamese (I am a cat.) <frank@any.net>"
2048-bit RSA key, ID CD8104F7, created 2017-10-27 (main key ID 4CFC6990)
gpg: gpg-agent is not available in this session
gpg: encrypted with 2048-bit RSA key, ID CD8104F7, created 2017-10-27
      "Frank Siamese (I am a cat.) <frank@any.net>"
This is TOP SECRET stuff that only Frank can see!!!!!
If anyone else see it, it's the end of the world as we know it.
(With apologies to REM.)
gpg: Signature made Fri 27 Oct 2017 06:39:15 PM EDT using RSA key ID
9FD7014B
gpg: Good signature from "Donald A. Tevault <donniet@something.net>"
frank@ubuntu:~$
```

10. With no more warning messages, this looks much better. At your end, do the same thing with Frank's public key.

5.4 – Signing a file without encryption

If a file isn't secret but you still need to ensure authenticity and integrity, you can just sign it without encrypting it:

1. Create an unencrypted message for Frank and then sign it:

```
donnie@ubuntu:~$ gpg -s not_secret_for_frank.txt
You need a passphrase to unlock the secret key for
user: "Donald A. Tevault <donniet@something.net>"
2048-bit RSA key, ID 9FD7014B, created 2017-10-27
gpg: gpg-agent is not available in this session
donnie@ubuntu:~$ ls -l
. . .
-rw-rw-r-- 1 donnie donnie  40 Oct 27 19:30 not_secret_for_frank.txt
-rw-rw-r-- 1 donnie donnie 381 Oct 27 19:31 not_secret_for_frank.txt.gpg
```

Just as before, this creates a .gpg version of the file.

2. Send the message to Frank.

3. Log in as Frank. Have him try to open it with less:

```
frank@ubuntu:~$ less not_secret_for_frank.txt.gpg
```

On older Linux distros, you'll see a lot of gibberish because of the signature, but you'll also see the plain-text message. On newer Linux distros, you'll only see the plain-text message, without the gibberish.

4. Have Frank use gpg with the --verify option to verify that the signature really does belong to you:

```
frank@ubuntu:~$ gpg --verify not_secret_for_frank.txt.gpg
gpg: Signature made Fri 27 Oct 2017 07:31:12 PM EDT using RSA key ID
9FD7014B
gpg: Good signature from "Donald A. Tevault <donniet@something.net>"
frank@ubuntu:~$
```

This wraps it up for our discussion of encrypting individual files. Let's now take a look at encrypting block devices and directories.

5.7 – Encrypting a home directory for a new user account

In Chapter 3, Securing Normal User Accounts, I showed you how Ubuntu allows you to encrypt a user's home directory as you create his or her user account. To review, let's see the command for creating Goldie's account:

1. If it hasn't already been done, install the ecryptfs-utils package:

```
sudo apt install ecryptfs-utils
```

2. On an Ubuntu VM, create Goldie's account with an encrypted directory:

```
sudo adduser --encrypt-home goldie
```

Have Goldie log in. Have her unwrap her mount passphrase, write it down, and store it in a secure place. She'll need it if she ever needs to recover a corrupted directory:

```
ecryptfs-unwrap-passphrase .ecryptfs/wrapped-passphrase
```

When you use adduser --encrypt-home , home directories for new users will automatically be set to a restrictive permissions value that will keep everyone out except for the owner of the directory. This happens even on Ubuntu 20.04 when you leave the adduser.conf file set with its default settings.

5.8 – Encrypting other directories with eCryptfs

Encrypting other directories is a simple matter of mounting them with the ecryptfs filesystem:

1. Create a secrets2 directory in the top level of the filesystem:

```
donnie@ubuntu2204-packt:~$ sudo mkdir /secrets2
[sudo] password for donnie:
donnie@ubuntu2204-packt:~$
```

2. Use mount with the -t ecryptfs option to encrypt the directory. Note that you'll list the directory name twice, because then it will be used as its own mount point. From the menu, choose 1 to enter your desired passphrase, and choose the encryption algorithm and the key length:

```
donnie@ubuntu2204-packt:~$ sudo mount -t ecryptfs /secrets2/ /secrets2/
Select key type to use for newly created files:
1) passphrase
2) tspi
Selection: 1
Passphrase:
Select cipher:
1) aes: blocksize = 16; min keysize = 16; max keysize = 32
2) blowfish: blocksize = 8; min keysize = 16; max keysize = 56
3) des3_ede: blocksize = 8; min keysize = 24; max keysize = 24
4) twofish: blocksize = 16; min keysize = 16; max keysize = 32
5) cast6: blocksize = 16; min keysize = 16; max keysize = 32
```

6) cast5: blocksize = 8; min keysize = 5; max keysize = 16

Selection [aes]:

Go with the default of aes , and 16 bytes for the key.

3. Go with the default of no for plaintext passthrough , and with yes for filename encryption:

Enable plaintext passthrough (y/n) [n]:

Enable filename encryption (y/n) [n]: y

4. Go with the default Filename Encryption Key and verify the mounting options:

Filename Encryption Key (FNEK) Signature [e339e1ebf3d58c36]:

Attempting to mount with the following options:

ecryptfs_unlink_sigs

ecryptfs_fnek_sig=e339e1ebf3d58c36

ecryptfs_key_bytes=16

ecryptfs_cipher=aes

ecryptfs_sig=e339e1ebf3d58c36

5. This warning only comes up when you mount the directory for the first time. For the final two questions, type yes in order to prevent that warning from coming up again:

WARNING: Based on the contents of [/root/.ecryptfs/sig-cache.txt],

it looks like you have never mounted with this key before. This could mean that you have typed yourpassphrase wrong.

Would you like to proceed with the mount (yes/no)? : yes

Would you like to append sig [e339e1ebf3d58c36] to

[/root/.ecryptfs/sig-cache.txt]

in order to avoid this warning in the future (yes/no)? : yes

Successfully appended new sig to user sig cache file

Mounted eCryptfs

6. Just for fun, create a file within your new encrypted secrets2 directory, and then unmount the directory. Then, try to do a directory listing:

cd /secrets2

sudo vim secret_stuff.txt

cd

```
sudo umount /secrets2
donnie@ubuntu2204-packt:~$ ls -l /secrets2/
total 12
-rw-rw-r-- 1 donnie donnie 12288 Oct 28 19:04 ECRYPTFS_FNEK_ENCRYPTED.
FXbXCS5fwxKABUQtEPlumGPaN-RGvqd13yybkpTr1eCVWVHdr-lrmi1X9Vu-mLM-A-
VeqIdN6KNZGcs-
donnie@ubuntu2204-packt:~$
```

By choosing to encrypt filenames, nobody can even tell what files you have when the directory is unmounted. When you're ready to access your encrypted files again, just remount the directory the same as you did before.

5.9 – Getting and installing VeraCrypt

Follow these steps to install VeraCrypt:

1. Download VeraCrypt from here: https://www.veracrypt.fr/en/Downloads.html

2. The Linux version of VeraCrypt comes two ways. First, there's a .tar.bz2 file, which contains a set of universal installer scripts that should work on any Linux distribution. Once you extract the .tar.bz2 archive file, you'll see three scripts for GUI installation and two for console-mode installation. There are scripts for both 32-bit and 64-bit versions of Linux:

```
donnie@donnie-VirtualBox:~$ tar xjvf veracrypt-1.25.9-setup.tar.bz2
veracrypt-1.25.9-setup-console-x64
veracrypt-1.25.9-setup-console-x86
veracrypt-1.25.9-setup-gtk3-console-x64
veracrypt-1.25.9-setup-gtk3-gui-x64
veracrypt-1.25.9-setup-gui-x64
veracrypt-1.25.9-setup-gui-x86
donnie@donnie-VirtualBox:~$
```

3. The executable permission is already set, so all you have to do to install is this:

```
donnie@donnie-VirtualBox:~$ ./veracrypt-1.25.9-setup-gui-x64
```

4. You'll need sudo privileges, but the installer will prompt you for your sudo password. After reading and agreeing to a rather lengthy license agreement, the installation only takes a few seconds.

5. Optional step: Recently, the VeraCrypt developers added a second way to install VeraCrypt on Linux. On their Downloads page, you'll now see that there are .deb and .rpm package files for various versions of Debian, Ubuntu, OpenSUSE, and Red Hat/CentOS/Fedora. For Debian/Ubuntu systems, install the package by doing:

sudo dpkg -i veracrypt-packageversion.deb

6. For Red Hat/CentOS/Fedora and OpenSUSE, do

sudo rpm -Uvh veracrypt-packageversion.rpm

7. Note though that there's still nothing here for the newer Red Hat 9-type distros, but I found that the CentOS 8 package does work on AlmaLinux 9.

8. End of lab.

5.10 – Creating and mounting a VeraCrypt volume in console mode

I haven't been able to find any documentation for the console-mode variant of VeraCrypt, but you can see a list of the available commands just by typing veracrypt . For this demo, you'll create a 2 GB encrypted directory. But you can just as easily create an encrypted directory elsewhere, such as on a USB memory stick:

1. To create a new encrypted volume, type the following:

veracrypt -c

2. This will take you into an easy-to-use interactive utility. For the most part, you'll be fine just accepting the default options:

```
donnie@ubuntu:~$ veracrypt -c
Volume type:
1) Normal
2) Hidden
Select [1]:
Enter volume path: /home/donnie/good_stuff
Enter volume size (sizeK/size[M]/sizeG): 2G
Encryption Algorithm:
1) AES
2) Serpent
. . .
```

. . .

Select [1]:

. . .

. . .

3. For the filesystem, the default option of FAT gives you the best cross-platform compatibility between Linux, macOS, and Windows:

Filesystem:

1) None

2) FAT

3) Linux Ext2

4) Linux Ext3

5) Linux Ext4

6) NTFS

7) exFAT

Select [2]:

4. Select your password and a PIM (short for Personal Iterations Multiplier). For my PIM, I entered 8891 . (High PIM values give better security, but they will also cause the volume to take longer to mount.) Then, type at least 320 random characters in order to generate the encryption key (this is where it would be handy to have my cats walking across my keyboard.):

Enter password:

Re-enter password:

Enter PIM: 8891

Enter keyfile path [none]:

Please type at least 320 randomly chosen characters and then press Enter:

5. After you hit the Enter key, be patient, because the final generation of your encrypted volume will take a few moments. Here, you see that my 2 GB good_stuff container has been successfully created:

donnie@ubuntu:~$ ls -l good_stuff

-rw------- 1 donnie donnie 2147483648 Nov 1 17:02 good_stuff

donnie@ubuntu:~$

6. Mount this container in order to use it. Begin by creating a mount point directory:

```
donnie@ubuntu:~$ mkdir good_stuff_dir
donnie@ubuntu:~$
```

7. Use the veracrypt utility to mount your container on this mount point:

```
donnie@ubuntu:~$ veracrypt good_stuff good_stuff_dir
Enter password for /home/donnie/good_stuff:
Enter PIM for /home/donnie/good_stuff: 8891
Enter keyfile [none]:
Protect hidden volume (if any)? (y=Yes/n=No) [No]:
Enter your user password or administrator password:
donnie@ubuntu:~$
```

8. To see what VeraCrypt volumes you have mounted, use veracrypt -l :

```
donnie@ubuntu:~$ veracrypt -l
1: /home/donnie/secret_stuff /dev/mapper/veracrypt1 /home/donnie/secret_
stuff_dir
2: /home/donnie/good_stuff /dev/mapper/veracrypt2 /home/donnie/good_
stuff_dir
donnie@ubuntu:~$
```

9. End of lab. That's all there is to it.

Lab Summary:

This lab section comprehensively covers essential aspects of Linux security strategies and applications. Starting with fundamental principles, it progresses through user authentication, file system security, network communication security, and encryption techniques. Through practical hands-on exercises, users gain insight into implementing robust security measures within a Linux environment. By grasping these concepts and applying them, individuals can enhance the security posture of their Linux systems, thereby mitigating potential risks and vulnerabilities.

LAB 6 – Access control lists and Implementing mandatory access control

Lab Objectives:

1. **Creating a Shared Group Directory**
    - Establish a group 'sales' and add users 'mimi', 'mrgray', and 'mommy' to the group.
    - Set up a shared directory '/sales', manage ownership, permissions, and implement Access Control Lists (ACLs) for users' file access.
    - Test user permissions within the directory and demonstrate ACL-based file restrictions.
2. **SELinux Type Enforcement**
    - Install Apache and SELinux tools.
    - Create a webpage, configure correct SELinux type for the content file, induce an SELinux violation, and rectify using 'restorecon'.
3. **SELinux Booleans and Ports**
    - Explore allowed ports for Apache under SELinux.
    - Modify Apache's listening port, observe SELinux errors, add and remove ports from the authorized list, and restore to default configuration.
4. **Troubleshooting an AppArmor Profile**
    - Utilize AppArmor utilities to troubleshoot Samba's enforcement mode.
    - Identify, edit, and reload AppArmor profiles for Samba, rectifying issues causing service restart failures.

8.1 – Creating a shared group directory

For this lab, you'll just put together everything that you've learned in this chapter to create a shared directory for a group. You can do this on any of your virtual machines:

1. On any virtual machine, create the sales group:

```
sudo groupadd sales
```

2. Create the users mimi, mrgray, and mommy, adding them to the sales group as you create the accounts.

On CentOS or AlamaLinux, do this:

```
sudo useradd -G sales mimi
sudo useradd -G sales mrgray
sudo useradd -G sales mommy
```

On Ubuntu, do this:

```
sudo useradd -m -d /home/mimi -s /bin/bash -G sales mimi
sudo useradd -m -d /home/mrgray -s /bin/bash -G sales mrgray
sudo useradd -m -d /home/mommy -s /bin/bash -G sales mommy
```

3. Assign each user a password.

4. Create the sales directory in the root level of the filesystem. Set proper ownership and permissions, including the SGID and sticky bits:

```
sudo mkdir /sales
sudo chown nobody:sales /sales
sudo chmod 3770 /sales
ls -ld /sales
```

5. Log in as Mimi, and have her create a file:

```
su - mimi
cd /sales
echo "This file belongs to Mimi." > mimi_file.txt
ls -l
```

6. Have Mimi set an ACL on her file, allowing only Mr. Gray to read it. Then, have Mimi log back out:

```
chmod 600 mimi_file.txt
setfacl -m u:mrgray:r mimi_file.txt
getfacl mimi_file.txt
ls -l
exit
```

7. Have Mr. Gray log in to see what he can do with Mimi's file. Then, have Mr. Gray create his own file and log back out:

```
su - mrgray
cd /sales
cat mimi_file.txt
echo "I want to add something to this file." >> mimi_file.txt
echo "Mr. Gray will now create his own file." > mr_gray_file.txt
```

ls -l

exit

8. Mommy will now log in and try to wreak havoc by snooping in other users' files and by trying to delete them:

su - mommy

cat mimi_file.txt

cat mr_gray_file.txt

rm -f mimi_file.txt

rm -f mr_gray_file.txt

exit

9. End of lab.

8.2 – SELinux type enforcement

In this lab, you'll install the Apache web server and the appropriate SELinux tools. You'll then view the effects of having the wrong SELinux type assigned to a web content file. If you're ready, let's go:

1. Install Apache, along with all the required SELinux tools on CentOS 7:

sudo yum install httpd setroubleshoot setools policycoreutils
policycoreutils-python

On AlmaLinux 8 or 9, use the following command:

sudo dnf install httpd setroubleshoot setools policycoreutils
policycoreutils-python-utils

2. Activate setroubleshoot by restarting the auditd service:

sudo service auditd restart

3. Enable and start the Apache service and open port 80 on the firewall:

sudo systemctl enable --now httpd
sudo firewall-cmd --permanent --add-service=http
sudo firewall-cmd --reload

4. In the /var/www/html/ directory, create an index.html file with the following contents:

```
<html>
  <head>
    <title>SELinux Test Page</title>
  </head>
  <body>
    This is a test of SELinux.
  </body>
</html>
```

5. View the information about the index.html file:

```
ls -Z index.html
```

6. In your host machine's web browser, navigate to the IP address of the CentOS virtual machine. You should be able to view the page.

7. Induce an SELinux violation by changing the type of the index.html file to something that's incorrect:

```
sudo chcon -t tmp_t index.html
ls -Z index.html
```

8. Go back to your host machine's web browser and reload the document. You should now see a Forbidden message.

9. Use restorecon to change the file back to its correct type:

```
sudo restorecon index.html
```

10. Reload the page in your host machine's web browser. You should now be able to view the page.

11. End of lab.

9.1 – SELinux Booleans and ports

In this lab, you'll view the effects of having Apache try to listen on an unauthorized port:

1. View the ports that SELinux allows the Apache web server daemon to use:

```
sudo semanage port -l | grep 'http'
```

2. Open the /etc/httpd/conf/httpd.conf file in your favorite text editor. Find the line that says Listen 80 and change it to Listen 82. Restart Apache by entering the following:

```
sudo systemctl restart httpd
```

3. View the error message you receive by entering:

```
sudo tail -20 /var/log/messages
```

4. Add port 82 to the list of authorized ports and restart Apache:

```
sudo semanage port -a 82 -t http_port_t -p tcp
sudo semanage port -l
sudo systemctl restart httpd
```

5. Delete the port that you just added:

```
sudo semanage -d 82 -t http_port_t -p tcp
```

6. Go back into the /etc/httpd/conf/httpd.conf file and change Listen 82 back to Listen 80.
Restart the Apache daemon to return to normal operation.

7. End of lab.

9.2 – Troubleshooting an AppArmor profile

Perform this lab on an Ubuntu 18.04 VM. Carry out the following steps for troubleshooting:

1. Install the AppArmor utilities and the extra profiles:

```
sudo apt install apparmor-utils apparmor-profiles apparmor-profiles-extra
```

2. Install Samba and verify that it's running:

```
sudo apt install samba
sudo systemctl status smbd
```

```
sudo systemctl status nmbd
```

3. Set the two aforementioned Samba policies to enforce mode and try to restart Samba:

```
cd /etc/apparmor.d
sudo aa-enforce /usr/sbin/smbd usr.sbin.smbd
sudo aa-enforce /usr/sbin/nmbd usr.sbin.nmbd
sudo systemctl restart smbd
```

4. Note that Samba should fail to restart. (It will take quite a while before it finally errors out, so be patient.)

5. Look in the /var/log/syslog file to see if you can spot the problem.

6. Edit the /etc/apparmor.d/usr.sbin.smbd file. In the capability stanza, add this line:

```
capability net_admin,
```

7. At the bottom of the rules sections, under the /var/spool/samba/** rw, line, add this line:

```
/run/systemd/notify rw,
```

8. Save the file and reload the policy:

```
sudo apparmor_parser -r usr.sbin.smbd
```

9. As before, try to restart the Samba service, and verify that it started properly:

```
sudo systemctl restart smbd
sudo systemctl status smbd
```

10. End of lab.

Lab Summary:

This section covered crucial aspects of access control, mandatory access control using SELinux, and troubleshooting AppArmor profiles, encompassing the following key activities:

- **Shared Group Directory Creation:** Established a shared directory for a user group, assigned proper permissions, and implemented ACLs to control user access within the shared space.
- **SELinux Type Enforcement:** Installed Apache, configured a webpage, and manipulated SELinux types to showcase access denial and restoration using 'restorecon'.
- **SELinux Booleans and Ports:** Explored Apache's allowed ports under SELinux, modified Apache's listening port, and managed authorized ports to comprehend SELinux behavior.

- **AppArmor Profile Troubleshooting:** Leveraged AppArmor utilities to troubleshoot Samba's enforcement mode by editing and reloading profiles to rectify service restart failures.

Through these exercises, you gained hands-on experience in setting up access control mechanisms, managing SELinux policies, and troubleshooting security profiles, enhancing your skills in access control and security management in Linux environments.

LAB 7 – Kernel hardening and process isolation

Lab Objectives:

1. **Kernel Parameter Scanning and Hardening**: Utilize Lynis to scan and assess kernel parameters, comprehend security implications, and harden the Linux kernel for improved system security.
2. **Setting Kernel Capabilities**: Enable a normal user to execute specific privileged tasks by setting kernel capabilities, as demonstrated by allowing a user to run a Python web server using Python 2 on an AlmaLinux system.
3. **Implementing Application Sandboxing with Firejail**: Understand and apply Firejail, an application sandboxing tool, to confine the operations of applications like web browsers and office suites within a controlled environment for enhanced security.

10.1 – Scanning kernel parameters with Lynis.

Lynis is in the normal repositories for Ubuntu and in the EPEL repository for CentOS and AlmaLinux. It's always a few versions behind what you can get directly from the author's website, but for now, that's okay. When we get to Chapter 14, Vulnerability Scanning and Intrusion Detection, I'll show you how to get the newest version. Let's get started:

1. Install Lynis from the repository for Ubuntu, like this:

```
sudo apt update
sudo apt install lynis
```

Do this for CentOS 07:

```
sudo yum install lynis
```

Do this for AlmaLinux 8 or 9:

```
sudo dnf install lynis
```

2. Scan the system by using the following command:

```
sudo lynis audit system
```

3. When the scan completes, scroll back up to the [+] Kernel Hardening section of the output. Copy and paste the sysctl key pairs into a text file. Save it as secure_values.conf in your own home directory. The contents of the file should look something like this:

```
- fs.protected_hardlinks (exp: 1) [ OK ]
```

- fs.protected_symlinks (exp: 1) [OK]
- fs.suid_dumpable (exp: 0) [OK]
- kernel.core_uses_pid (exp: 1) [OK]
- kernel.ctrl-alt-del (exp: 0) [OK]
- kernel.dmesg_restrict (exp: 1) [DIFFERENT]
- kernel.kptr_restrict (exp: 2) [DIFFERENT]
- kernel.randomize_va_space (exp: 2) [OK]
- kernel.sysrq (exp: 0) [DIFFERENT]
- kernel.yama.ptrace_scope (exp: 1 2 3) [DIFFERENT]
- net.ipv4.conf.all.accept_redirects (exp: 0) [DIFFERENT]
- net.ipv4.conf.all.accept_source_route (exp: 0) [OK]

. . .

. . .

4. Use grep to send all of the DIFFERENT lines to a new file. Name it 60-secure_values.conf:

```
grep 'DIFFERENT' secure_values.conf > 60-secure_values.conf
```

5. Edit the 60-secure_values.conf file to convert it into the sysctl configuration format. Set each parameter to the exp value that's currently within the pairs of parentheses. The finished product should look something like this:

```
kernel.dmesg_restrict = 1
kernel.kptr_restrict = 2
kernel.sysrq = 0
kernel.yama.ptrace_scope = 1 2 3
net.ipv4.conf.all.accept_redirects = 0
net.ipv4.conf.all.log_martians = 1
net.ipv4.conf.all.send_redirects = 0
net.ipv4.conf.default.accept_redirects = 0
net.ipv4.conf.default.log_martians = 1
net.ipv6.conf.all.accept_redirects = 0
net.ipv6.conf.default.accept_redirects = 0
```

6. Copy the file to the /etc/sysctl.d/ directory:

```
sudo cp 60-secure_values.conf /etc/sysctl.d/
```

7. Reboot the machine to read in the values from the new file:

```
sudo shutdown -r now
```

8. Repeat step 2. Most items should now show up with their most secure values. However, you might see a few DIFFERENT lines come up. That's okay; just move the lines for those parameters into the main /etc/sysctl.conf file and reboot the machine again.

That's the end of the lab—congratulations!

10.2 – Setting a kernel capability

For this lab, you'll allow a normal user to run a Python web server. You'll need to use Python 2 for this, which isn't in the newest Linux distros. (Python 3 won't work.) So for this one, use your AlmaLinux 8 VM. Let's get started:

1. If Apache is installed on your virtual machine, ensure that it's stopped:

```
sudo systemctl stop httpd
```

2. Install Python 2:

```
sudo dnf install python2
```

3. From within your own home directory, attempt to start the Python SimpleHTTPServer with just your normal user privileges, and note the error message:

```
python2 -m SimpleHTTPServer 80
```

4. See if any capabilities are set on the Python executable file:

```
getcap /usr/bin/python2.7
```

5. Set the CAP_NET_BIND_SERVICE capability on the Python executable file:

```
sudo setcap 'CAP_NET_BIND_SERVICE+ep' /usr/bin/python2.7
```

6. Repeat steps 3 and 4. This time, it should work.

7. Ensure that port 80 is open on the virtual machine firewall and use your host machine's web browser to access the server.

8. Shut down the web server using Ctrl + C.

9. View the capabilities that have been assigned to the ping executable:

```
getcap /usr/bin/ping
```

10. Review the capabilities of the man page, especially the part about the various capabilities that are there

10.3 – Using Firejail

For this lab, you'll use an AlmaLinux 9 virtual machine with the Gnome desktop. Let's get started:

1. Create an AlmaLinux virtual machine with the Gnome desktop option.

2. Install the EPEL repository and update the VM with these commands:

```
sudo dnf install epel-release
sudo dnf upgrade
```

Then, reboot the machine.

3. Install Firejail, LibreOffice, and Chromium:

```
sudo dnf install firejail libreoffice-x11 chromium
```

4. In one terminal window, start Chromium without any kernel capabilities:

```
firejail --caps.drop=all chromium-browser
```

5. Surf to various websites to see if everything works as it should.

6. In another terminal window, start LibreOffice, also without any capabilities:

```
firejail --caps.drop=all libreoffice
```

7. Create the various types of LibreOffice documents and try out various LibreOffice functions to see how much still works properly.

8. Shut down both Chromium and LibreOffice.

9. Configure Firejail so that it automatically sandboxes every application you start, even if you do this from the normal Start menu:

```
sudo firecfg
```

10. Look at the symbolic links that were created:

```
ls -l /usr/local/bin
```

11. Try to open Firefox from the normal menu and verify whether or not it works. Then, shut down Firefox.

12. To be able to run Firefox without Firejail, just delete its symbolic link from the /user/local/bin/ directory, like so:

```
sudo rm /usr/local/bin/firefox
```

13. Try to run Firefox again. You should see that it starts normally.

Lab Summary:

This lab section extensively covers various aspects of Linux kernel hardening, process isolation, and application sandboxing. It begins with employing Lynis to scan and secure kernel parameters, emphasizing the importance of configuring secure values for improved system security. Additionally, it demonstrates the process of setting kernel capabilities to grant specific privileges to users while maintaining system integrity.

Furthermore, the lab explores Firejail as a tool for sandboxing applications, providing a controlled environment for programs like web browsers and office suites. Through practical exercises, users gain hands-on experience in implementing sandboxing techniques, evaluating the impact on application functionalities, and configuring Firejail for automatic application sandboxing. Overall, these exercises reinforce crucial concepts related to enhancing Linux security by hardening the kernel, managing kernel capabilities, and implementing application-level isolation mechanisms.

LAB 8 – Applying Best Practices for Secure Software Management

Lab Objectives:

1. **Updating Debian-based Systems**: Learn the step-by-step process to update and manage packages on Debian-based systems such as Ubuntu. Understand how to update the package lists, perform system upgrades, remove old packages, and configure automatic updates.
2. **Configuring Auto Updates for Ubuntu**: Explore the configuration of automatic updates on Ubuntu systems, including enabling/disabling automatic reboots after updates, setting update times, and managing various update types.
3. **Updating Red Hat 7-based Systems**: Gain proficiency in updating Red Hat 7-based systems (e.g., CentOS) by executing commands to upgrade the system, check for security-related updates, configure automatic updates, and understand the needs-restarting command.
4. **Updating Red Hat 8/9-based Systems with DNF**: Acquire knowledge on updating Red Hat 8/9-based systems using the DNF package manager. Learn how to upgrade the system, set up automatic updates using **dnf-automatic**, and use the **needs-restarting** command for system reboot status verification.

1.6 – Updating Debian-based systems

Step 1: On Debian and its many children, including Ubuntu, run two commands, as shown here:

```
sudo apt update
sudo apt dist-upgrade
```

Step 2: Occasionally, you'll also need to remove some old packages that are no longer needed. How will you know? Easy. When you log in to the system, a message will appear on the command line. To remove these old packages, just run this command:

```
sudo apt auto-remove
```

Next, we will configure auto updates for Ubuntu

Configuring auto updates for Ubuntu

When you first install Ubuntu 22.04, automatic updates are turned on by default. To verify that, you'll first check the status of the unattended-upgrades service, like so:

```
donnie@ubuntu2204-packt:~$ systemctl status unattended-upgrades
● unattended-upgrades.service - Unattended Upgrades Shutdown
Loaded: loaded (/lib/systemd/system/unattended-upgrades.service; enabled;
vendor preset: enabled)
Active: active (running) since Sat 2022-10-08 19:25:54 UTC; 52min ago
. . .
```

```
. . .
donnie@ubuntu2204-packt:~$
```

Then, look in the /etc/apt/apt.conf.d/20auto-upgrades file. If auto-updating is enabled, you'll see this:

```
APT::Periodic::Update-Package-Lists "1";
APT::Periodic::Unattended-Upgrade "1";
```

I must confess, though, that I have mixed feelings about this. I mean, it's nice that the security updates get installed without me having to do anything, but a lot of those updates require that the system be rebooted before they can take effect. By default, Ubuntu systems don't automatically reboot after an update is installed. If you keep it that way, you'll see a message about it when you log into the system. But if you prefer, you can set Ubuntu to automatically reboot after it automatically updates itself.

Here's how to do it:

Step 1: Go into the /etc/apt/apt.conf.d directory and open the 50unattended-upgrades file in your favorite text editor. In the vicinity of line 67, you'll see a line that says:

```
//Unattended-Upgrade::Automatic-Reboot "false";
```

Step 2: Uncomment the line by removing the leading slashes, and change false to true , like so:

```
Unattended-Upgrade::Automatic-Reboot "true";
```

Step 3: With this new configuration, Ubuntu will now reboot itself immediately after the automatic update process has completed. If you'd rather have the machine reboot at a specific time, scroll down to about line 103, where you'll see this:

```
//Unattended-Upgrade::Automatic-Reboot-Time "02:00";
```

Step 4: Since this line is commented out with its pair of leading slashes, it currently has no effect. To have the machine reboot at 2:00 A.M., just uncomment this line. To have it reboot at, say, 10:00 P.M., uncomment the line and change the time to 22:00 , like so:

```
Unattended-Upgrade::Automatic-Reboot-Time "22:00";
```

Of course, there's that old, basic precept that thou shalt not install system updates on a production system without first testing them on a test system. Any operating system vendor can occasionally supply you with problematic updates, and that has included Ubuntu. (I know what you're saying: Preach it, Donnie.) Ubuntu's automatic update feature is in direct opposition to that basic precept. If automatic updates have been enabled, disabling them is quite easy, if you choose to do so.

Step 5: To disable automatic updates, just go into the /etc/apt/apt.conf.d directory and open the 20auto-upgrades file in your favorite text editor. Here's what you'll see:

```
APT::Periodic::Update-Package-Lists "1";
APT::Periodic::Unattended-Upgrade "1";
```

Step 6: Change the parameter for that second line to 0 , so that the file will now look like this:

```
APT::Periodic::Update-Package-Lists "1";
APT::Periodic::Unattended-Upgrade "0";
```

Now, the system will still check for updates and show a message on the login screen when any are available, but it won't automatically install them. And of course, it should go without saying that you need to check your systems on a regular basis to see if updates are available. If you do prefer to leave automatic updates enabled, be sure to either enable automatic rebooting or to log in to the system at least a couple of times a week to see if it needs to be rebooted.

Step 7: If you want to see if there are any security-related updates available, but don't want to see any non-security updates, use the unattended-upgrade command, like so:

```
sudo unattended-upgrade --dry-run -d
```

Step 8: To manually install the security-related updates without installing non-security updates, just run:

```
sudo unattended-upgrade -d
```

If you're running some form of desktop Ubuntu on a workstation that gets shut down after every use, you can enable the automatic updates if you like, but there's no need to enable automatic reboots.

Also, if you're running a non-Ubuntu flavor of Debian, which would include Raspbian for the Raspberry Pi, you can give it the same functionality as Ubuntu by installing the unattended-upgrades package. Just run this command:

sudo apt install unattended-upgrades

You can also use the apt command to install only the security updates, but it would require piping the apt output into a convoluted set of text filters in order to mask the non-security updates. Using the unattended-upgrade command is much easier.

1.7 – Updating Red Hat 7-based systems

With Red Hat-based systems, which include CentOS and Oracle Linux, there's no automatic update mechanism that you can set up during installation. So, with the default configuration, you'll need to perform updates yourself:

Step 1: To update a Red Hat 7-based system, just run this one command:

```
sudo yum upgrade
```

Step 2: Sometimes, you might just want to see if there are any security-related updates that are ready to be installed. Do that by running this command:

```
sudo yum updateinfo list updates security
```

Step 3: If any security updates are available, you'll see them at the end of the command output. On the system that I just tested, there was only one security update available, which looks like this:

```
FEDORA-EPEL-2019-d661b588d2 Low/Sec. nagios-common-4.4.3-1.el7.x86_64
updateinfo list done
```

Step 4: If the only thing you want to install is the security updates, run this command:

```
sudo yum upgrade --security
```

Step 5: Now, let's say that you need a CentOS system to automatically update itself. You're in luck because there's a package for that. Install and enable it, and start it by running these two commands:

```
sudo yum install yum-cron
sudo systemctl enable --now yum-cron
```

Step 6: To configure it, go into the /etc/yum directory, and edit the yum-cron.conf file. At the top of the file, you'll see this:

```
[commands]
# What kind of update to use:
# default = yum upgrade
# security = yum --security upgrade
# security-severity:Critical = yum --sec-severity=Critical upgrade
# minimal = yum --bugfix update-minimal
# minimal-security = yum --security update-minimal
# minimal-security-severity:Critical = --sec-severity=Critical update-
minimal
update_cmd = default
```

This lists the various types of upgrades we can do. The last line shows that we're set to update everything.

Step 7: Let's say that you only want security updates to get applied automatically. Just change the last line to the following:

```
update_cmd = security
```

Step 8: On lines 15 and 20, you'll see this:

```
download_updates = yes
apply_updates = no
```

Step 9: This indicates that by default, yum-cron is only set to automatically download updates, but not to install them.

Step 10: If you want the updates to get automatically installed, change the apply_updates parameter to yes .

Note that unlike Ubuntu, there's no setting to make the system automatically reboot itself after an update.

Step 11: Finally, let's look at the mail settings for yum-cron , which you'll find on lines 48 through 57 of the yum-cron.conf file, as shown here:

```
[email]
# The address to send email messages from.
# NOTE: 'localhost' will be replaced with the value of system_name.
email_from = root@localhost
# List of addresses to send messages to.
email_to = root
# Name of the host to connect to to send email messages.
email_host = localhost
```

As you can see, the email_to = line is set to send messages to the root user account. If you want to receive messages on your own account, just change it here.

Step 12: To see the messages, you'll need to install a mail reader program, if one isn't already installed. (It hasn't been installed if you chose Minimal installation when you installed the operating system.) Your best bet is to install mutt, like so:

```
sudo yum install mutt
```

Step 13: When you open mutt and look at a message, you'll see something like this:

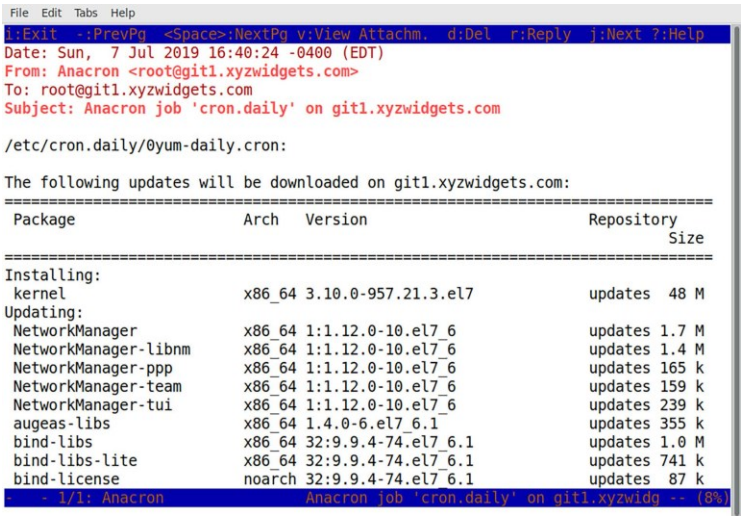


Figure 1.16: Mutt mail client

Step 14: As with all operating systems, certain updates will require that the system be restarted. And how do you know when the system needs to be restarted? With the needs-restarting command, of course. First, though, you need to make sure that needs-restarting is installed on your system. Do that with this command:

```
sudo yum install yum-utils
```

Once the package is installed, there are three ways to use needs-restarting . If you just run the command without any option switches, you'll see the services that need to be restarted and the packages that require you to reboot the machine. You can also use the -s or -r options, as shown here:

Command	Explanation
`sudo needs-restarting`	This shows the services that need to be restarted, and the reasons why the system might need to be rebooted.
`sudo needs-restarting -s`	This only shows the services that need to be restarted.
`sudo needs-restarting -r`	This only shows the reasons why the system needs to be rebooted.

1.8 – Updating Red Hat 8/9-based systems

The old yum utility has been around for practically forever, and it's been a good, hard-working utility. But it does have its occasional quirks, and at times it can be excruciatingly slow. But, not to worry. Our heroes at Red Hat have finally done something about that, by replacing yum with dnf . So, when you work with your AlmaLinux 8/9 virtual machines, you'll use dnf instead of yum . Let's see how to do this:

Step 1: For the most part, you use dnf the same way that you'd use yum , with the same arguments and options. For example, to do a system upgrade, just do:

```
sudo dnf upgrade
```

Step 2: The main functional difference between yum and dnf is that dnf has a different automatic update mechanism. Instead of installing the yum-cron package, you'll now install the dnf-automatic package, like so:

```
sudo dnf install dnf-automatic
```

Step 3: In the /etc/dnf directory, you'll see the automatic.conf file, which you'll configure the same way as you did the yum-cron.conf file for CentOS 7. Instead of working as a cron job, as the old yum-cron did, dnf-automatic works with a systemd timer. When you first install dnf-automatic, the timer is disabled. Enable it and start it by running this command:

```
sudo systemctl enable --now dnf-automatic.timer
```

Step 4: Verify that it's running with this command:

```
sudo systemctl status dnf-automatic.timer
```

Step 5: If it started successfully, you should see something like this when you check the status:

```
[donnie@redhat-8 ~]$ sudo systemctl status dnf-automatic.timer
dnf-automatic.timer - dnf-automatic timer
```

```
Loaded: loaded (/usr/lib/systemd/system/dnf-automatic.timer; enabled;
vendor preset: disabled)
Active: active (waiting) since Sun 2019-07-07 19:17:14 EDT; 13s ago
Trigger: Sun 2019-07-07 19:54:49 EDT; 37min left
Jul 07 19:17:14 redhat-8 systemd[1]: Started dnf-automatic timer.
[donnie@redhat-8 ~]$
```

To determine if a system needs to be restarted, just install the yum-utils package and run the needs-restarting command, the same as you did for CentOS 7. (For some reason, the Red Hat developers never bothered to change the package name to dnf-utils.) For more details about dnf-automatic, just type:

man dnf-automatic

Lab Summary:

This lab section provides comprehensive guidance on updating and managing software packages on Linux systems of various distributions. It starts with instructions on updating Debian-based systems, including Ubuntu, by executing essential commands like **apt update**, **apt dist-upgrade**, and **apt autoremove** to maintain system integrity.
Furthermore, the lab demonstrates the configuration of automatic updates for Ubuntu, allowing users to explore enabling/disabling automatic reboots, setting specific update times, and managing security and non-security updates through configuration file modifications.

Moving on to Red Hat-based systems, the lab covers updating Red Hat 7-based systems by executing **yum upgrade**, checking for security updates, configuring automatic updates using **yum-cron**, and understanding the **needs-restarting** command to identify necessary system reboots.

For Red Hat 8/9-based systems, the lab introduces the usage of **dnf** for system upgrades and the setup of automatic updates using **dnf-automatic**. It walks through the configuration process, enabling and verifying the **dnf-automatic** timer, and utilizing **dnf-utils** to check for system reboot requirements.

Overall, the lab equips users with the essential skills to update, manage, and configure software packages efficiently on Debian-based systems, Red Hat 7-based systems, and newer Red Hat 8/9-based systems using appropriate package managers and update mechanisms available in each Linux distribution.

LAB 9 – Loging, auditing, and hardening the server

Lab Objectives:

1. **Installing Logwatch**: Learn to set up Logwatch along with necessary components like mutt and Postfix on different Linux distributions (Ubuntu, CentOS, AlmaLinux). Configure Logwatch to deliver daily log summaries and modify the level of detail in the log reports.
2. **Setting up a Basic Log Server**: Understand the process of configuring a log-collecting server using rsyslog on Ubuntu, CentOS, and AlmaLinux. Set up clients to forward syslog messages to the log server and verify the log collection from both the server and client VMs.
3. **Installing ClamAV and Maldet**: Install ClamAV antivirus and Linux Malware Detect (maldet) on various Linux distributions (Ubuntu, CentOS, AlmaLinux). Configure ClamAV and maldet to protect against malware and scan for potential threats.
4. **Configuring Maldet**: Explore the configuration of maldet by modifying monitoring paths, enabling email alerts, configuring quarantine settings, and starting the maldet daemon to monitor specified directories for potential malware.
5. **Using Auditd**: Practice using the auditd tool by setting up audit rules, monitoring file changes, creating user accounts, changing directory permissions, and analyzing audit logs to track user activities and file access.
6. **Using Pre-configured Rules with Auditd**: Simulate adherence to Security Technical Implementation Guides (STIG) auditing standards by implementing pre-configured rulesets provided by auditd. Understand how to delete custom rules, use pre-existing rulesets, and restart the auditd service to meet compliance requirements.

11.1 – Installing Logwatch

To deliver its messages, Logwatch requires that the machine also has a running mail server daemon. Depending on the options you chose when installing the operating system, you might or might not already have the Postfix mail server installed. When Postfix is set up as a local server, it will deliver system messages to the root user's local account. To view the Logwatch summaries on the local machine, you'll also need to install a text-mode mail reader, such as mutt.

For this lab, you can use any of your VMs:

1. Install Logwatch, mutt, and Postfix. (On Ubuntu, choose the local option when installing Postfix. With CentOS or AlmaLinux, the local option is already the default.) For Ubuntu, do this:

```
sudo apt install postfix mutt logwatch
```

For CentOS 7, do this:

```
sudo yum install postfix mutt logwatch
```

For AlmaLinux, do this:

sudo dnf install postfix mutt logwatch

2. On Ubuntu only, create a mail spool file for your user account:

sudo touch /var/mail/your_user_name

3. Open the /etc/aliases file in your favorite text editor. Configure it to forward the root user's mail to your own normal account by adding the following line at the bottom of the file:

root: your_user_name

4. Save the file, and then copy the information from it to a binary file that the system can read. Do that with this:

sudo newaliases

5. At this point, you have a fully operational implementation of Logwatch that will deliver daily log summaries with a low level of detail. To see the default configuration, look at the default configuration file:

less /usr/share/logwatch/default.conf/logwatch.conf

6. To change the configuration, edit the /etc/logwatch/conf/logwatch.conf file on CentOS and AlmaLinux, or create the file on Ubuntu. Change to a medium level of logging detail by adding this line:

Detail = Med

7. Perform some actions that will generate some log entries. You can do that by performing a system update, installing some software packages, and using sudo fdisk -l to view the partition configuration.

8. If possible, allow your VM to run overnight. In the morning, view your log summary by doing this:

mutt

9. When prompted to create a Mail directory in your home directory, hit the Y key.

10. End of lab.

11.2 – Setting up a basic log server

Setting up the server is identical on Ubuntu, CentOS, and AlmaLinux. There's only one minor difference in setting up the clients. For best results, ensure that the server VM and the client VM each have a different hostname:

1. On the log-collecting server VM, open the /etc/rsyslog.conf file in your favorite text editor and look for these lines, which are near the top of the file:

```
# Provides TCP syslog reception
#module(load="imtcp") # needs to be done just once
#input(type="imtcp" port="514")
```

2. Uncomment the bottom two lines and save the file. The stanza should now look like this:

```
# Provides TCP syslog reception
module(load="imtcp") # needs to be done just once
input(type="imtcp" port="514")
```

3. Restart the rsyslog daemon:

```
sudo systemctl restart rsyslog
```

4. If the machine has an active firewall, open port 514/tcp.

5. Next, configure the client machines. For Ubuntu, add the following line to the bottom of the /etc/rsyslog.conf file, substituting the IP address of your own server VM:

```
@@192.168.0.161:514
```

6. For CentOS and AlmaLinux, look for this stanza at the bottom of the /etc/rsyslog.conf file:

```
# ### sample forwarding rule ###
#action(type="omfwd"
# An on-disk queue is created for this action. If the remote host is
# down, messages are spooled to disk and sent when it is up again.
#queue.filename="fwdRule1"       # unique name prefix for spool files
#queue.maxdiskspace="1g"        # 1gb space limit (use as much as
possible)
#queue.saveonshutdown="on"      # save messages to disk on shutdown
#queue.type="LinkedList"        # run asynchronously
#action.resumeRetryCount="-1"    # infinite retries if host is down
# Remote Logging (we use TCP for reliable delivery)
# remote_host is: name/ip, e.g. 192.168.0.1, port optional e.g. 10514
#Target="remote_host" Port="XXX" Protocol="tcp"
```

Remove the comment symbols from each line that isn't obviously a real comment. Add the IP address and port number for the log server VM. The finished product should look like this:

```
# ### sample forwarding rule ###
action(type="omfwd"
# An on-disk queue is created for this action. If the remote host is
# down, messages are spooled to disk and sent when it is up again.
queue.filename="fwdRule1"       # unique name prefix for spool files
queue.maxdiskspace="1g"         # 1gb space limit (use as much as
possible)
queue.saveonshutdown="on"       # save messages to disk on shutdown
queue.type="LinkedList"         # run asynchronously
action.resumeRetryCount="-1"    # infinite retries if host is down
# Remote Logging (we use TCP for reliable delivery)
# remote_host is: name/ip, e.g. 192.168.0.1, port optional e.g. 10514
Target="192.168.0.161" Port="514" Protocol="tcp")
```

7. Save the file and then restart the rsyslog daemon.

8. On the server VM, verify that messages from both the server VM and the client VM are getting sent to the log files. (You can tell by the different hostnames for different messages.)

9. This is the end of the lab.

12.1 – Installing ClamAV and maldet

We'll begin by installing ClamAV. (It's in the normal repository for Ubuntu, but not for CentOS or AlmaLinux. For CentOS and AlmaLinux, you'll need to install the EPEL repository, as I showed you in Chapter 1, Running Linux in a Virtual Environment.) We'll also install wget, which we'll use to download LMD. For this lab, you can use Ubuntu, CentOS 7, or either version of AlmaLinux. Let's get started:

1. The following command will install ClamAV, inotify-tools and wget on Ubuntu:

```
donnie@ubuntu3:~$ sudo apt install clamav wget inotify-tools
```

The following command will install ClamAV, inotify-tools, and wget on CentOS 7:

```
[donnie@localhost ~]$ sudo yum install clamav clamav-update wget inotifytools
```

For AlmaLinux 8 or AlmaLinux 9, do this:

```
[donnie@localhost ~]$ sudo dnf install clamav clamav-update wget inotifytools
[donnie@localhost ~]$ sudo systemctl enable –now clamav-freshclam
```

Note that if you chose the Minimal installation option when creating a CentOS or an AlmaLinux virtual machine (VM), you may also have to install the perl and the tar packages.

For Ubuntu, the clamav package contains everything you need. For CentOS or AlmaLinux, you'll need to also install clamav-update in order to obtain virus updates.

The rest of the steps will be the same for either VM.

2. Next, you'll download and install LMD. Here, you'll want to do something that I rarely tell people to do. That is, you'll want to log in to the root user shell. The reason is that, although the LMD installer works fine with sudo, you'll end up with the program files being owned by the user who performed the installation, instead of by the root user. Performing the installation from the root user's shell saves us the trouble of tracking down those files and changing the ownership. So, download the file as follows:

```
sudo su wget
http://www.rfxn.com/downloads/maldetect-current.tar.gz
```

Now, you'll have the file in the root user's home directory.

3. Extract the archive and enter the resultant directory:

```
tar xzvf maldetect-current.tar.gz
cd maldetect-1.6.4/
```

4. Run the installer. Once the installer finishes, copy the README file to your own home directory so that you can have it for ready reference. (This README file is the documentation for LMD.) Then, exit from the root user's shell back to your own shell:

```
root@ubuntu3:~/maldetect-1.6.4# ./install.sh
Created symlink from /etc/systemd/system/multi-user.target.wants/maldet.
service to /usr/lib/systemd/system/maldet.service.
update-rc.d: error: initscript does not exist: /etc/init.d/maldet
. . .
. . .
maldet(22138): {sigup} signature set update completed
maldet(22138): {sigup} 15218 signatures (12485 MD5 | 1954 HEX | 779 YARA
| 0 USER)
root@ubuntu3:~/maldetect-1.6.4# cp README /home/donnie
root@ubuntu3:~/maldetect-1.6.4# exit
logout
donnie@ubuntu3:~$
```

As you see, the installer automatically creates the symbolic link that enables the maldet service, and it also automatically downloads and installs the newest malware signatures.

5. For CentOS or AlmaLinux, the maldet.service file that the installer copied to the /lib/systemd/system/ directory has the wrong SELinux context, which will prevent maldet from starting. Correct the SELinux context like this:

```
sudo restorecon /lib/systemd/system/maldet.service
```

6. You've reached the end of the lab – congratulations!

12.2 – Configuring maldet

In previous versions, maldet was configured by default to automatically monitor and scan users' home directories. In its current version, the default is for it to only monitor the /dev/shm/, /var/tmp/, and /tmp/ directories. We're going to reconfigure it so that we can add some directories. Let's get started:

1. Open the /usr/local/maldetect/conf.maldet file for editing. Find these two lines:

```
default_monitor_mode="users"
```

default_monitor_mode="/usr/local/maldetect/monitor_paths"

2. Change them to look like this:

# default_monitor_mode="users"
default_monitor_mode="/usr/local/maldetect/monitor_paths"

3. At the top of the file, enable email alerts and set your username as the email address. The two lines should now look something like this:

email_alert="1"
email_addr="donnie"

4. LMD isn't already configured to move suspicious files to the quarantine folder, and we want to make it do that. Further down in the conf.maldet file, look for the line that says:

quarantine_hits="0"

5. Change it to this:

quarantine_hits="1"

You'll see a few other quarantine actions that you can configure, but, for now, this is all we need.

6. Save the conf.maldet file, because that's all the changes that we need to make to it.

7. Open the /usr/local/maldetect/monitor_paths file for editing. Add the directories that you want to monitor, like this:

/var/tmp
/tmp
/home
/root

Since viruses affect Windows and not Linux, just monitor the directories with files that will be shared with Windows machines.

8. After you save the file, start the maldet daemon:

sudo systemctl start maldet

You can add more directories to the monitor_paths file at any time, but remember to restart the maldet daemon any time that you do, in order to read in the new additions.

12.4 – Using auditd

In this lab, you'll practice using the features of auditd. Let's get started:

1. For Ubuntu only, install auditd:

```
sudo apt update
sudo apt install auditd
```

2. View the rules that are currently in effect:

```
sudo auditctl -l
```

3. From the command line, create a temporary rule that audits the /etc/passwd file for changes. Verify that the rule is in effect:

```
sudo auditctl -w /etc/passwd -p wa -k passwd_changes
sudo auditctl -l
```

4. Create a user account for Lionel. On Ubuntu, do this:

```
sudo adduser lionel
```

On CentOS or AlmaLinux, do this:

```
sudo useradd lionel
sudo passwd lionel
```

5. Search for audit messages regarding any changes to the passwd file:

```
sudo ausearch -i -k passwd_changes
sudo aureport -i -k | grep 'passwd_changes'
```

6. Log out of your own account and log in as Lionel. Then, log out of Lionel's account and back in to your own.

7. Do an authentication report:

```
sudo aureport -au
```

8. Create the /secrets directory and set the permissions so that only the root user can access it:

```
sudo mkdir /secrets
sudo chmod 700 /secrets
```

9. Create a rule that monitors the /secrets directory:

```
sudo auditctl -w /secrets -k secrets_watch
```

```
sudo auditctl -l
```

10. Log out of your account, and log in as Lionel. Have him try to view what's in the /secrets directory:

```
ls -l /secrets
```

11. Log out of Lionel's account and log in to your own. View the alerts that Lionel created:

```
sudo ausearch -i -k secrets_watch | less
```

12. You now have two temporary rules that will disappear when you reboot the machine. Make them permanent by creating a custom.rules file:

```
sudo sh -c "auditctl -l > /etc/audit/rules.d/custom.rules"
```

13. Reboot the machine and verify that the rules are still in effect:

```
sudo auditctl -l
```

You've reached the end of the lab – congratulations!

12.5 – Using pre-configured rules with auditd

In this lab, we'll simulate that the US government is our client, and that we need to set up a server that will meet their Security Technical Implementation Guides (STIG) auditing standards. To do that, we'll use several pre-configured rulesets that get installed when you install auditd. Note that this lab will work on any of your virtual machines:
Delete the custom.rules file that you created in the previous lab, and then restart the auditd service.
Copy the 10-base-config.rules, 30-stig.rules, 31-privileged.rules, and 99-finalize.rules files to the /etc/audit/rules.d/ directory. (These rules files are in the /usr/share/doc/auditd/examples/rules/ directory on Ubuntu, and in the /usr/share/audit/sample-rules/ directory on AlmaLinx.):

```
[donnie@almalinux9 sample-rules]$ pwd
/usr/share/audit/sample-rules
[donnie@almalinux9 sample-rules]$ sudo cp 10-base-config.rules 30-stig.rules
31-privileged.rules 99-finalize.rules /etc/audit/rules.d
[donnie@almalinux9 sample-rules]$
```

Restart the auditd service, and then use sudo auditctl -l to view the new active ruleset.

End of lab.

Lab Summary:

This section of labs aimed at enhancing security measures within a Linux environment covered diverse aspects:

- **Log Management and Monitoring:** Implemented Logwatch for log analysis, configured a basic log server, and set up ClamAV and LMD for malware detection.
- **System Auditing and Hardening:** Utilized auditd to monitor file changes, user activities, and directory access while adhering to STIG compliance standards through pre-configured rulesets. By understanding and executing these procedures, you've gained practical experience in fortifying Linux systems, bolstering defenses against potential threats, and ensuring compliance with security standards.

LAB 10 – VULNERABILITY SCANNING AND INTRUSION DETECTION

Lab Objectives:

1. **Installing Snort via a Docker Container**
    - Utilize Podman to install Snort in a Docker container on Ubuntu, AlmaLinux 8, or AlmaLinux 9.
    - Validate Snort configuration, download and transfer rule sets, and analyze a sample .pcap file.
    - Explore Snort tutorial videos for further understanding.
2. **Creating an IPFire Virtual Machine**
    - Set up a virtual machine with IPFire, configuring two network interfaces (Bridged and NAT).
    - Enable Intrusion Prevention System (IPS) within IPFire, select and customize rulesets, and enable IPS for specified interfaces.
    - Explore IPS logs and navigate through other features within the IPFire dashboard.
3. **Installing Nikto from GitHub**
    - Clone and set up Nikto repository on Kali, Debian, or Ubuntu, ensuring necessary Perl modules are available.
    - Execute Nikto commands, update signature databases, and explore basic functionality.

13.1 – Installing Snort via a Docker container

You'll definitely want to go with the container option instead of the source code option. That's because the directions for setting up the source code option aren't as clear as they should be, and one particular library package doesn't always compile properly. Instead of using the official Docker software, I'll be showing you how to use Podman, which is Red Hat's drop-in replacement for Docker. Podman's security is better than that of Docker, and it's available for pretty much every Linux distro. Podman is already installed on your AlmaLinux 8 and 9 virtual machines, but you'll need to install it yourself on Ubuntu.

1. On Ubuntu only, install the podman package:

```
sudo apt update
sudo apt install podman
```

2. On Ubuntu only, open the /etc/containers/registries.conf file in your text editor. Find this line:

```
# unqualified-search-registries = ["example.com"]
```

3. Change it to this:

unqualified-search-registries = ["docker.io"]

4. Download and start the container:

podman run --name snort3 -h snort3 -u snorty -w /home/snorty -d -it
ciscotalos/snort3 bash

5. Next, enter the container so that you can interact with the Snort commands:

podman exec -it snort3 bash

6. If this command executes successfully, you'll find yourself at the snorty@snort3 command prompt.

7. Validate the Snort configuration with this single-word command:

snort

8. Snort requires a set of rules that define the potential problems that it should analyze. Paying customers will receive up-to-date rulesets, while non-paying users can download rulesets that are about one month behind. An old ruleset from 2018 comes with the Docker container, so you'll want something that's a bit more recent. You won't be able to download the rulesets directly to your container, so you'll need to download them to either your virtual machine or to your host machine, and then transfer them to the container. On either your host machine or in another terminal that's connected to the virtual machine, download the latest community ruleset, like this:

wget https://www.snort.org/downloads/community/snort3-community-rules.tar.gz

9. You can't use scp or sftp to connect to the container from the virtual machine or your host machine, but you can use them to connect to the virtual machine or host machine from the container. So, from within the container, use sftp to transfer in the new ruleset file. Your commands should look something like this:

sftp donnie@192.168.0.20
get snort3-community-rules.tar.gz
bye

10. While still within the container, unarchive the ruleset file and transfer the new ruleset to its proper location:

snort3-community-rules.tar.gz
cd snort3-community-rules
cp snort3-community.rules ~/snort3/etc/rules/

11. Test things out by examining a .pcap file that's included with the example files:

snort -q --talos --rule-path snort3/etc/rules/ -r examples/intro/lab2/

eternalblue.pcap

12. Take a look at the tutorial videos at the Snort website. You can see them here: https://www.snort.org/resources

13. When you're done, type exit to get out of the container. To shutdown the container, do this:

podman kill snort3

14. End of lab.

13.2 – Creating an IPFire virtual machine

You won't normally run IPFire in a virtual machine. Instead, you'll install it on a physical machine that has at least two network interfaces. But, just for the sake of letting you see what it looks like, setting it up in a virtual machine will do for now. Let's get started:

1. Create a virtual machine with two network interfaces. Set one to Bridged mode and leave the other in NAT mode. Install IPFire into this virtual machine. During the setup portion, select the Bridged adapter as the Green interface and select the NAT adapter as the Red interface.

2. After you install IPFire, you'll need to use the web browser of your normal workstation to navigate to the IPFire dashboard. Do this with this URL:https://192.168.0.190:444

(Of course, substitute your own IP address for your Green interface.)

3. Under the Firewall menu, you'll see an entry for Intrusion Prevention. Click on that to get to this screen, where you can enable Intrusion Prevention. The first step for that is to click on the Add provider button that's under the Ruleset Settings section.

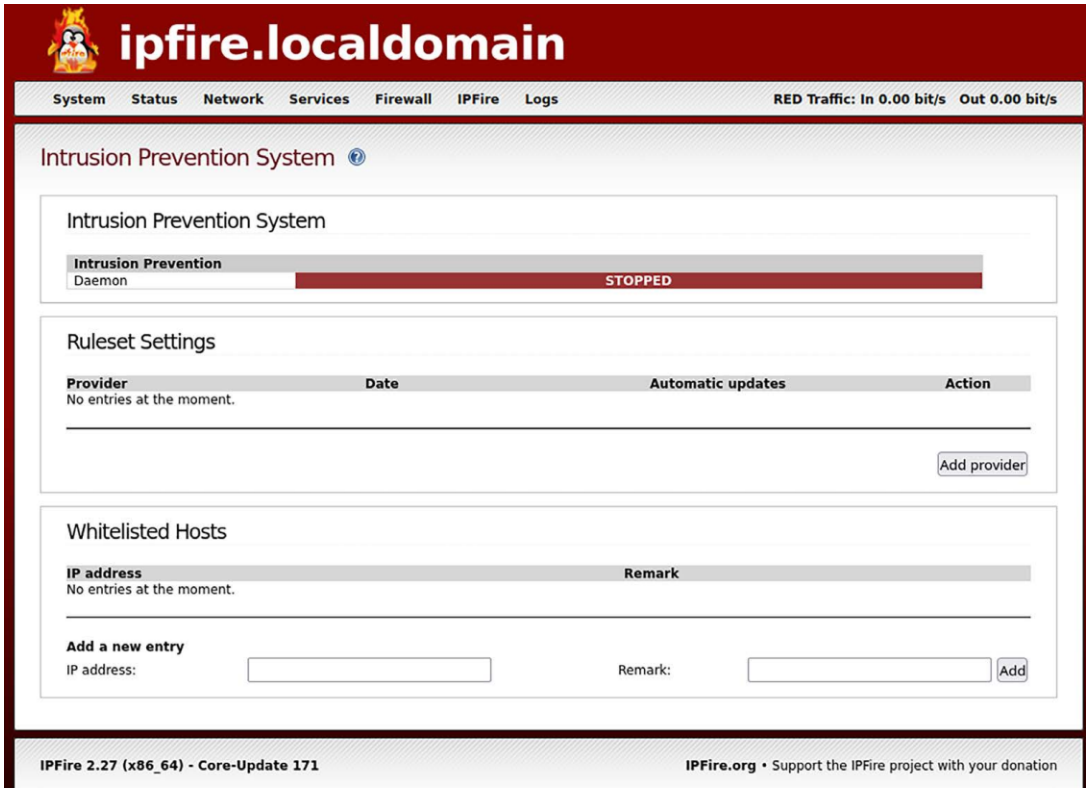


Figure 14.3: Click the Add provider button

4. On the next page, select the ruleset that you want to use. Leave the Enable automatic updates checkbox enabled. Then, hit the Add button:

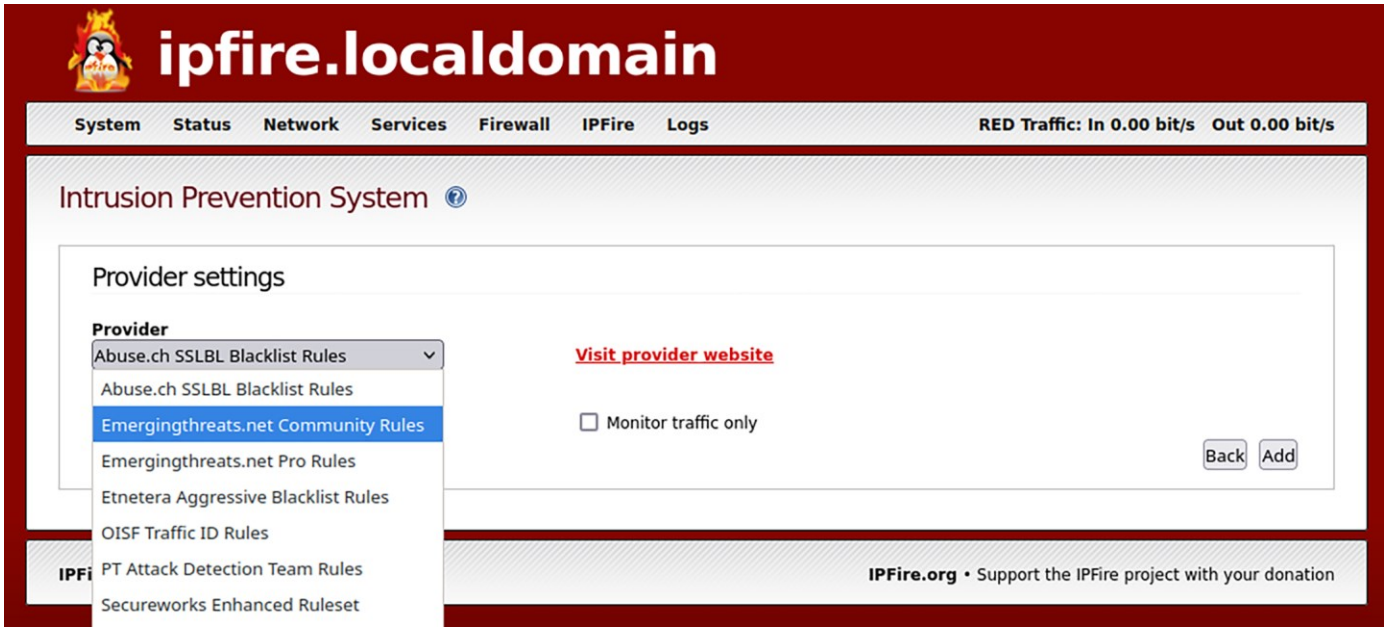


Figure 14.4: Select the ruleset

5. You will then see this screen, where you'll select the interfaces for which you want to enable intrusion revention. (Select both interfaces.) Then, select the Enable Intrusion Prevention System checkbox and click on Save:

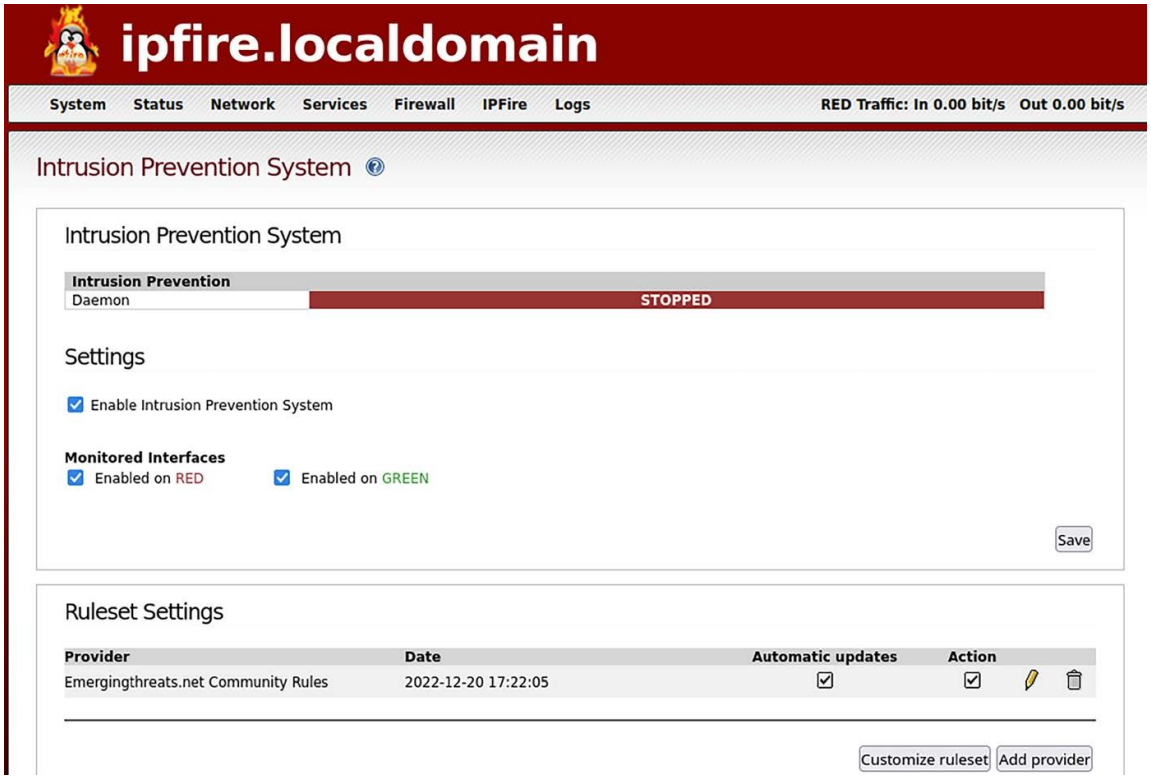


Figure 14.5: Enable the IPS

If all goes well, you'll see the following output:

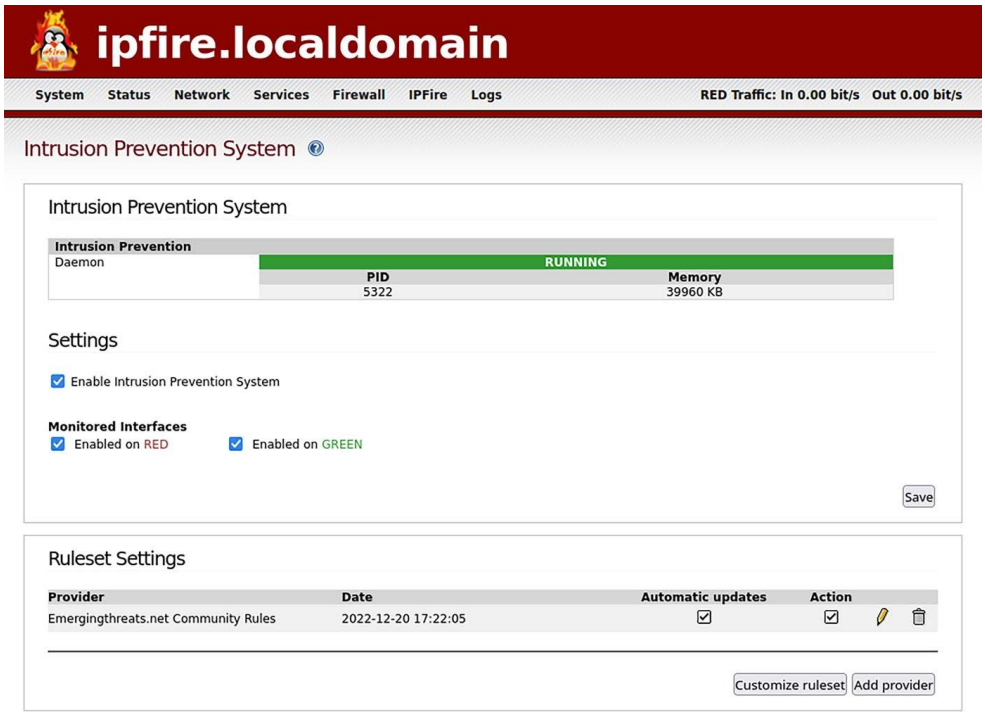


Figure 14.6: With the IPS enabled

6. In the Ruleset Settings section, click the Customize ruleset button. On the next page, click on the rules that you want to enable. Then, at the bottom of the screen, click the Apply button:

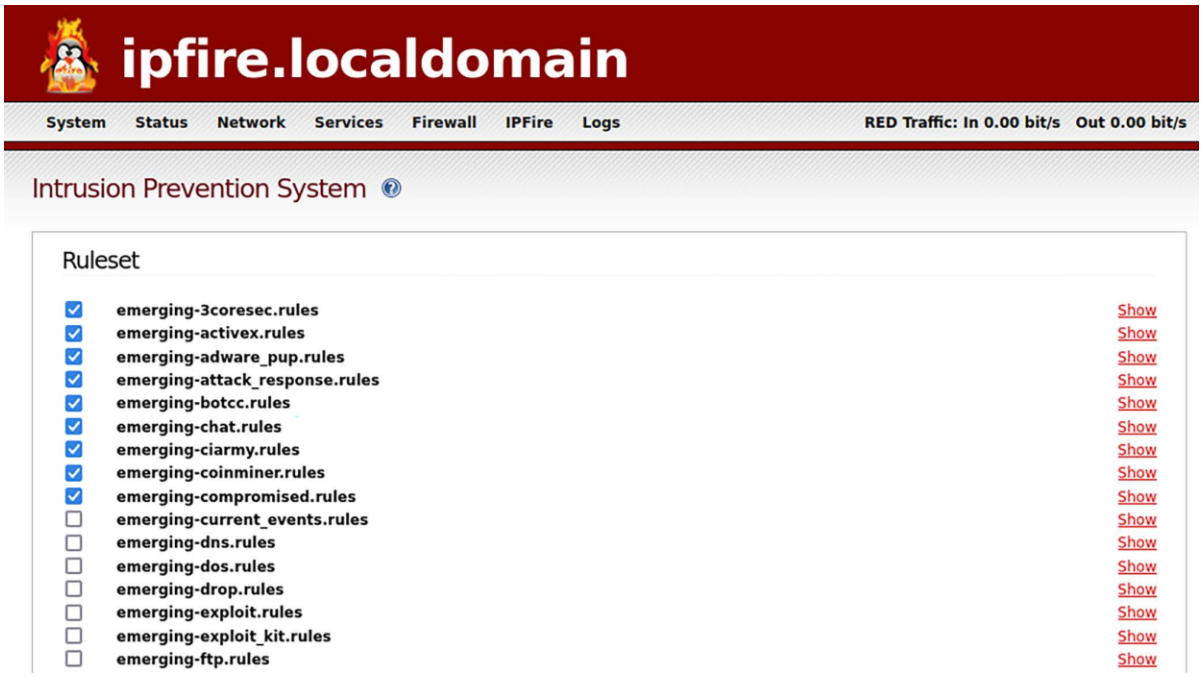


Figure 14.7: Select the desired rules

7. View what's going on with the IPS by selecting Log/IPS Logs. (Note that what you see will depend upon which rules that you've chosen to enable. Even then, it might take a while for any entries to show up.)

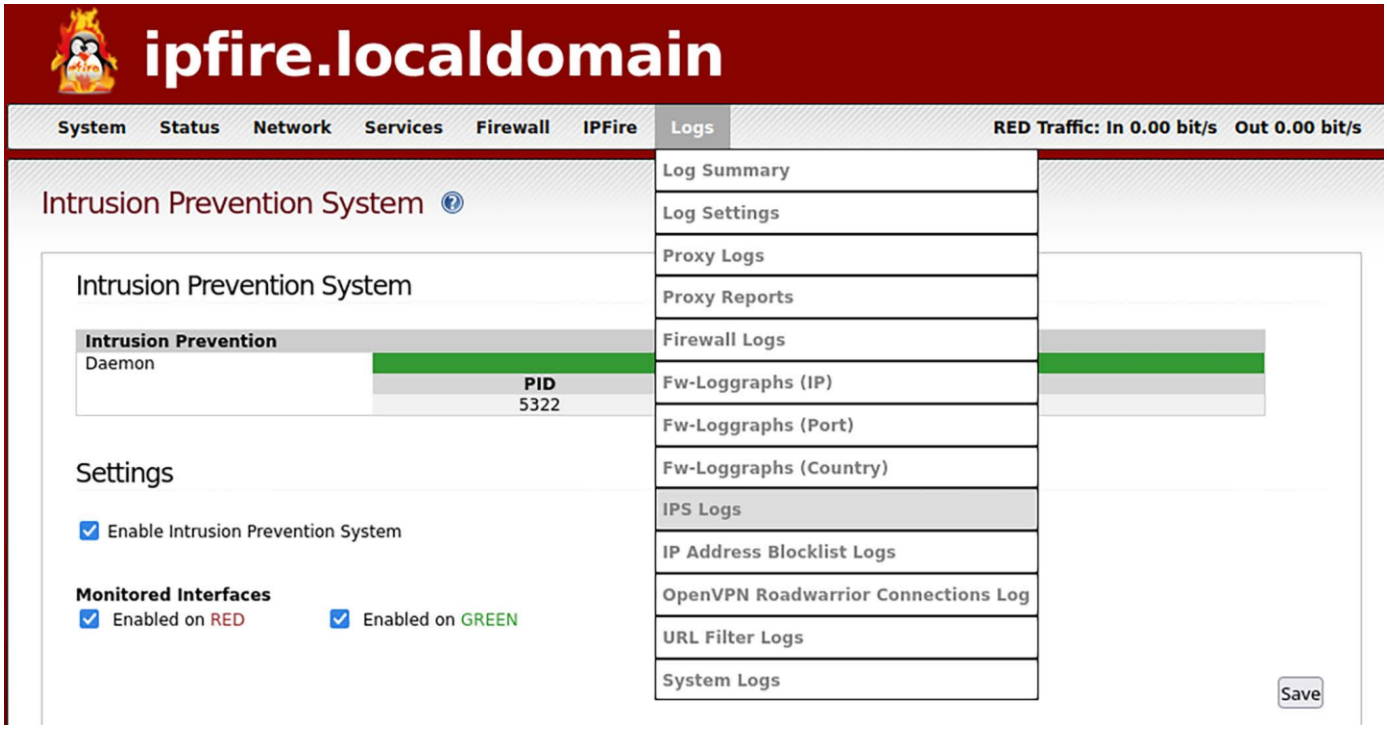


Figure 14.8: Look at the IPS logs

8. Click on the other menu items to view IPFire's other features.

13.3 – Installing Nikto from Github

To make things easy, we'll do this on Kali, because it already has all of the perl modules that Nikto needs to operate. If you do this on Debian or Ubuntu, it should work, but you'll need to chase down the perl modules that it needs yourself. And, forget about doing this on AlmaLinux, because the necessary perl modules aren't even in any of the AlmaLinux or EPEL repositories. (There's an alternate way to install them, but that's beyond the scope of this book.)

1. In your normal user home directory, clone the Nikto repository. Then, cd into the nikto directory, and check out the current branch:

```
git clone https://github.com/sullo/nikto.git
cd nikto
git checkout nikto-2.5.0
```

2. To run Nikto, cd into the program subdirectory, and invoke Nikto from there. For example, to see the Nikto help screen, do this:

```
cd program
```

./nikto -help

3. Periodically, you'll want to update the Nikto signature databases. Just cd into the nikto directory and do:

git pull

4. End of lab.

Lab Summary:

This section delved into key aspects of vulnerability scanning and intrusion detection, encompassing the following:

- **Snort Implementation:** Installed Snort within a Docker container using Podman, configured rule sets, and examined sample network traffic for intrusion detection.
- **IPFire Setup:** Created a virtual machine running IPFire, configured IPS, selected rulesets, and explored IPS logs for intrusion detection insights.
- **Nikto Installation:** Deployed Nikto from GitHub, set up required Perl modules, executed commands, and managed database updates for vulnerability scanning. By completing these tasks, you've gained practical experience in deploying intrusion detection systems like Snort, configuring IPS rulesets, and using vulnerability scanning tools like Nikto, enhancing your skills in identifying and preventing potential threats within a networked environment.