Lab: Environment Setup

Introduction:

Linux is an open-source operating system (OS). An operating system is the software that directly manages a system's hardware and resources, like CPU, memory, and storage. The OS sits between applications and hardware and makes the connections between all of your software and the physical resources that do the work.

The Linux kernel boasts an array of built-in security defences including **firewalls** that use packet filters in the kernel, the UEFI Secure Boot firmware verification mechanism, the Linux Kernel Lockdown configuration option and the **SELinux** or AppArmor Mandatory Access Control (MAC) security enhancement systems.

By enabling these features and configuring them to provide the highest level of security in a practice known as Linux kernel self-protection, administrators can add an additional layer of security to their systems.

But in this Lab exercise we will be relaxing some of the security features to ensure the Linux Security features doesn't become a barrier learning Docker.

Objectives:

- **Disable SELinux**
- **Disable firewalled**
- **Update and Reboot the System**

1. For ease of learning Docker, Disable **SELinux** in this training lab environment.

Security-Enhanced Linux (**SELinux**) is a mandatory access control (MAC) security mechanism implemented in the kernel.

SELinux has three basic modes of operation, of which Enforcing is set as the installation default mode.

- **Enforcing:** The default mode which will enable and enforce the SELinux security policy on the system, denying access and logging actions
- **Permissive:** In Permissive mode, SELinux is enabled but will not enforce the security policy, only warn and log actions. Permissive mode is useful for troubleshooting SELinux issues.
- **Disabled:** SELinux is turned off

To disable the **SELinux** run the below commands.

Note: Ensure you have logged in as **root** user with the password as **linux**.

```
# sed -i 's/enforcing/disabled/g' /etc/selinux/config
```

```
# setenforce 0
```

```
# sestatus
```

Output:

```
[root@docker-engine ~]#sestatus
SELinux status:                 enabled
SELinuxfs mount:                /sys/fs/selinux
SELinux root directory:         /etc/selinux
Loaded policy name:             targeted
Current mode:                   permissive
Mode from config file:          disabled
Policy MLS status:              enabled
Policy deny_unknown status:     allowed
Memory protection checking:     actual (secure)
Max kernel policy version:      33
```

2. Let's disable **Firewall** service in this training lab environment.

FirewallD is a frontend controller for iptables used to implement the persistent network traffic rules.

- Working with FirewallD has two main differences compared to directly controlling iptables:
- FirewallD uses zones and services instead of chain and rules. It manages rulesets dynamically, allowing updates without breaking existing sessions and connections.

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

Output:

```
[root@docker-engine ~]#systemctl disable --now firewalld
Removed "/etc/systemd/system/multi-user.target.wants/firewalld.service".
Removed "/etc/systemd/system/dbus-org.fedoraproject.FirewallD1.service".
```

```
# systemctl status firewalld --no-pager
```

Output:

```
[root@docker-engine ~]#systemctl status firewalld --no-pager
○ firewalld.service - firewalld - dynamic firewall daemon
     Loaded: loaded (/usr/lib/systemd/system/firewalld.service; disabled; vendor preset: enabled)
     Active: inactive (dead)
       Docs: man:firewalld(1)

Feb 04 15:34:23 localhost systemd[1]: Starting firewalld - dynamic firewall daemon...
Feb 04 15:34:24 localhost systemd[1]: Started firewalld - dynamic firewall daemon.
Feb 04 15:50:17 docker-engine systemd[1]: Stopping firewalld - dynamic firewall daemon...
Feb 04 15:50:17 docker-engine systemd[1]: firewalld.service: Deactivated successfully.
Feb 04 15:50:17 docker-engine systemd[1]: Stopped firewalld - dynamic firewall daemon.
Feb 04 15:50:17 docker-engine systemd[1]: firewalld.service: Consumed 2.205s CPU time.
```

3. Let's **update** the system with the latest packages and reboot the host to take effect.

```
# dnf update -y
```

Output:

```
Installed:
  grub2-tools-efi-1:2.06-55.el9.x86_64 grub2-tools-extra-1:2.06-55.el9.x86_64 kernel-5.14.0-247.el9.x86_64
  kernel-core-5.14.0-247.el9.x86_64    kernel-modules-5.14.0-247.el9.x86_64   libatomic-11.3.1-4.3.el9.x86_64
  python3-distro-1.5.0-7.el9.noarch

Complete!
```

3.1 After **rebooting** login again with **root** user.

```
# reboot
```