**Lab Exercise 9- Understanding and Enhancing Docker Security**

**Objective:**

To understand basic Docker security concepts and implement container security best practices, including user permissions, container capabilities, securing Docker networks, and more.

**Prerequisites:**

* Docker installed on your system
* Basic understanding of Docker containers, images, and networks
* Root or administrator access on your system

**Part 1: Running Containers as Non-Root Users**

1. **Default Behavior of Docker Containers:**

By default, Docker containers run as the root user, which can pose security risks. Let’s start by running a container as root:

docker run -it ubuntu:20.04 bash

Inside the container, check the user by running:

whoami

You will see that the container is running as root.

1. **Creating a Non-Root User in the Container:**

Exit the container and create a new Dockerfile to add a non-root user.

mkdir docker-security && cd docker-security

nano Dockerfile

Add the following to your Dockerfile:

FROM ubuntu:20.04

# Create a new user with a home directory

RUN useradd -ms /bin/bash appuser

# Switch to the new user

USER appuser

# Set the default working directory and command

WORKDIR /home/appuser

CMD ["bash"]

1. **Build and Run the Image:**

Build the Docker image:

docker build -t secure-container .

Run the container:

docker run -it secure-container

Check the current user again by running:

whoami

You should see appuser, meaning the container is now running as a non-root user.

1. **Verify Permissions:**

Try running a command that requires root privileges, such as installing a package:

apt-get update

You should get a permission denied error, which confirms that the user does not have root access.

**Part 2: Limiting Container Capabilities**

Docker containers come with a wide set of capabilities, some of which may not be necessary for certain workloads and could expose security vulnerabilities.

1. **Running a Container with Full Capabilities:**

Run a container and list the capabilities it has:

docker run --rm -it ubuntu:20.04 capsh --print

This shows a list of Linux capabilities assigned to the container. By default, Docker containers come with several capabilities that may not be needed for basic tasks.

1. **Restricting Capabilities:**

You can restrict the capabilities granted to the container using the --cap-drop flag. For example, drop the NET\_ADMIN and SYS\_ADMIN capabilities:

docker run --rm -it --cap-drop=NET\_ADMIN --cap-drop=SYS\_ADMIN ubuntu:20.04

Now, inside the container, try to perform network administration tasks, such as:

ifconfig

You should receive an error because the NET\_ADMIN capability has been removed.

1. **Adding Specific Capabilities:**

You can add back only the capabilities you need using the --cap-add flag. For example, add back the NET\_ADMIN capability:

docker run --rm -it --cap-add=NET\_ADMIN ubuntu:20.04 bash

Now, inside the container, try the same network administration command again. It should now work because you re-added the capability.

**Part 3: Securing Docker Networks**

Securing communication between containers is critical for Docker security. By default, Docker containers can communicate with each other on the same network, which might not always be desirable.

1. **Running Containers on an Isolated Network:**

Docker’s bridge network allows containers to communicate with each other by default. Let’s create an isolated custom network:

docker network create isolated-net

Now run two containers on this isolated network:

docker run -dit --name container1 --network isolated-net ubuntu:20.04

docker run -dit --name container2 --network isolated-net ubuntu:20.04

1. **Test Communication Between Containers:**

Get into one of the containers and try to ping the other:

docker exec -it container1 bash

ping container2

The containers should be able to communicate since they are on the same custom network.

1. **Using a none Network for Isolation:**

If you want to completely isolate a container from the network, you can use the none network mode:

docker run -dit --name isolated-container --network none ubuntu:20.04

This container will not be able to communicate with other containers or the internet.

**Part 5: Scanning Docker Images for Vulnerabilities**

Docker images can contain security vulnerabilities, which is why it's important to scan images before deploying them.

1. **Using docker scan:**

Docker provides a built-in vulnerability scanning tool called docker scan. Use it to scan an image for vulnerabilities:

docker scan ubuntu:20.04

The scan will return a list of any known vulnerabilities in the image, along with severity levels.

1. **Using Third-Party Scanners:**

Alternatively, you can use third-party scanning tools like **Trivy** or **Clair** for vulnerability scanning. For example, install **Trivy** and scan an image:

trivy image ubuntu:20.04

Review the results and consider updating the base image or applying security patches if vulnerabilities are found.

**Part 6: Using Docker Secrets for Sensitive Information**

Storing sensitive information (like passwords or API keys) securely is important when working with Docker.

1. **Create a Secret in Docker:**

Docker Swarm provides the ability to securely manage secrets. Let’s create a Docker secret:

echo "my\_secret\_password" | docker secret create db\_password -

1. **Use Secrets in a Service:**

You can use this secret in a Docker service. For example, create a simple service and pass the secret to it:

docker service create --name my\_secure\_service --secret db\_password alpine cat /run/secrets/db\_password

1. **Access the Secret Inside the Container:**

The secret will be available inside the container at /run/secrets/db\_password. Docker ensures that secrets are securely managed and not exposed.

**Conclusion:**

This lab exercise provides hands-on practice in securing Docker containers by:

* Running containers as non-root users
* Limiting container capabilities
* Isolating containers on custom networks
* Protecting the Docker daemon socket
* Scanning images for vulnerabilities
* Managing secrets securely in Docker

By applying these security best practices, you can significantly reduce the attack surface of your Docker environments.