

# Scan Docker images for vulnerabilities and implement security best practices

---

## Table of Contents

---

- [Description](#)
- [Problem Statement](#)
- [Prerequisites](#)
  - [Software Requirement](#)
  - [Hardware Requirement](#)
- [Trivy Installation Steps](#)
- [Implementation Steps](#)
  - [Step-1: Scan Docker Images for Vulnerabilities](#)
  - [Step-2: Implement Docker Security Best Practices](#)
- [References](#)

## Description

---

This guide explains how to secure Docker images and containers by scanning for vulnerabilities and applying security best practices. We'll use tools like **Docker Scan** and **Trivy** to detect vulnerabilities and suggest best practices for improving the security posture of Dockerized applications.

## Problem Statement

---

Security vulnerabilities in Docker images can lead to serious risks in production environments. It's essential to scan images and follow best practices to minimize these risks. This guide will help you secure the **Java TodoApp** and **MySQL** containers by identifying vulnerabilities and hardening Docker security.

## Prerequisites

---

Completion of all previous lab guides (up to Lab Guide-08) is required before proceeding with Lab Guide-09.

### Software Requirement

- **Docker Desktop**: Installed on your Windows machine.
- **Trivy**: A vulnerability scanner for containers.
- **TodoAPP\_MYSQL**: To download the source folder [click here](#)


### Hardware Requirement

- Minimum of 4 GB RAM
- At least 2 cores in the processor
- 5 GB of free storage space for Docker images and security tools

# Trivy Installation Steps

---

## 1. Download the **Trivy binary for Windows**

- **Install Trivy** if you haven't already. You can download Trivy for Windows from the [official GitHub repository](#).
- In the README file, click on download binary from <https://github.com/aquasecurity/trivy/releases/latest/>
- Download the appropriate Windows binary archive (**trivy\_XXXX\_Windows-64bit.zip**). 

## 2. Unpack the archive

- Locate the downloaded **.zip** file and extract it using an archive tool like **WinRAR** or **7-Zip**.
- This will extract a folder containing the **trivy.exe** binary.

## 3. Add Trivy to your PATH

To use Trivy from any command prompt, add the folder where **trivy.exe** is located to your system's PATH:

- Right-click on **This PC** (or **My Computer**) and select **Properties**.
- Click on **Advanced system settings** on the left.
- In the **System Properties** window, click **Environment Variables**.
- Under **System variables**, find the **Path** variable, select it, and click **Edit**.
- Click **New** and paste the path where **trivy.exe** is located (e.g., **C:\path\to\trivy\folder**).
- Click **OK** to close all the windows.



## 4. Verify the installation

Open **Command Prompt** and type the following to check if Trivy is installed correctly:

```
trivy --version
```

You should see the Trivy version printed, confirming the installation is successful.



## 5. Set Execution Permissions (if necessary)

By default, Windows executables do not require explicit permission to run. However, ensure that the file is not blocked:

- Right-click on **trivy.exe**, go to **Properties**, and check if there is an "Unblock" button at the bottom of the window. If it exists, click **Unblock**, then **OK**.

# Implementation Steps

---

## Step-1: Scan Docker Images for Vulnerabilities

### 1.1 Pull the Docker image

1. **Pull the image** you want to scan (e.g., MySQL or Java-based TodoApp):

```
docker pull openjdk:11-jdk-slim
docker pull mysql:8.0
```



2. **Scan the Docker image** using the built-in Docker Scan command:

```
docker scan openjdk:11-jdk-slim
```

### 1.2 Using Trivy (Recommended)

1. **Install Trivy** if you haven't already. You can download Trivy for Windows from the [official GitHub repository](#).
2. **Scan an image** for vulnerabilities:

```
trivy image mysql:8.0
```

Trivy will display a detailed report of vulnerabilities, categorized by severity (low, medium, high, critical).



3. **Fix or mitigate vulnerabilities** based on the scan results. Update to the latest image versions or patch specific vulnerabilities as necessary.

### 1.3 Updating Vulnerable Images

If vulnerabilities are found, ensure you update the Docker images regularly. To pull the latest image versions:

```
docker pull mysql:latest
docker pull openjdk:latest
```

## Step-2: Implement Docker Security Best Practices

## 2.1 Use Official and Verified Base Images

1. **Choose secure, official base images** from trusted sources, like the official **MySQL** or **OpenJDK** images:

```
FROM openjdk:11-jdk-slim
```

2. **Specify exact versions** for the base image to prevent using unstable or vulnerable versions:

```
FROM openjdk:11-jdk-slim@sha256:<specific_sha>
```

Example:

```
FROM openjdk:11-jdk-slim@sha256:  
<868a4f2151d38ba6a09870cec584346a5edc8e9b71fde275eb2e0625273e2fd8>
```

## 2.2 Minimize Image Size

1. **Use smaller base images** such as **alpine** versions when possible to reduce the attack surface:

```
FROM openjdk:11-jdk-alpine
```

- **jdk-alpine**: Refers to an image based on **Alpine Linux**, a very lightweight Linux distribution.
- **Alpine Linux** is a security-oriented, lightweight Linux distribution. It is often used in Docker images because it is:
  - **Minimal in size**: The Alpine-based image is significantly smaller (usually around 5 MB) compared to other distributions like Ubuntu or Debian. This reduces the size of your Docker image, which can lead to faster build and deployment times.
  - **Efficient**: Alpine uses the apk package manager, which is optimized for small and fast installations.
  - **Security-focused**: Alpine is designed to be more secure, with security features like stack-smashing protection and other hardening features built in.

2. **Remove unnecessary files and layers** in the Dockerfile:

```
RUN apt-get update && apt-get install -y \  
    some-package && \  
    rm -rf /var/lib/apt/lists/*
```

This removes unnecessary files after installation.

## 2.3 Run Containers as Non-Root Users

1. By default, Docker containers run as **root**, which is a security risk. To mitigate this, create and use a non-root user inside the container:

```
RUN groupadd -r appgroup && useradd -r -g appgroup appuser  
USER appuser
```

2. This will ensure the application runs under a less-privileged user, limiting the damage if an attacker gains access to the container.

## 2.4 Limit Resource Usage with Docker Flags

1. **Limit CPU and memory usage** for containers to prevent resource exhaustion attacks:

```
docker run -d --name todoapp_container --memory="512m" --cpus="1.0" todoapp
```

This command restricts the container to use no more than 512 MB of RAM and 1 CPU core.

 LimitCPU

 LimitCPU1

## 2.5 Use Read-Only Filesystems

1. Make the container's filesystem read-only to prevent malicious code from modifying container files:

```
docker run --read-only todoapp
```

This prevents unwanted changes to the container's filesystem.

 ReadOnly

## 2.6 Limit Container Capabilities

1. **Drop unnecessary Linux capabilities** from the container, limiting its privileges:

```
docker run --network=todoapp_network -e MYSQL_HOST=mysqlldb --cap-drop=ALL --  
cap-add=NET_BIND_SERVICE todoapp
```

This command drops all unnecessary capabilities, only allowing the container to bind to network ports.

Note: Make sure mysqldb is up and running in the same network as todoapp before running the above command



## 2.7 Network Security

1. **Isolate containers on a custom Docker network** and minimize exposure to external networks:

```
networks:
  todoapp_network:
    driver: bridge
```

2. **Use Docker secrets** to store sensitive information like passwords, instead of environment variables.

## References

- Docker Documentation: <https://docs.docker.com/>
- Trivy: <https://github.com/aquasecurity/trivy>
- Docker Security Best Practices: <https://docs.docker.com/engine/security/>