Enhancing Docker Container Security by Restricting Capabilities and Configuring User Namespaces

Table of Contents

Description

This guide explains how to improve the security of Docker containers by **restricting container capabilities** and configuring **user namespaces**. Containers, by default, run with a broad set of Linux capabilities and as the root user. Reducing privileges and mapping container users to non-root host users adds an extra layer of security.

Problem Statement

Running Docker containers with excessive capabilities or as the root user can lead to security vulnerabilities, especially if a container is compromised. This document provides steps to:

- Restrict unnecessary capabilities.
- Map container users to non-root users on the host system using **user namespaces**.

Prerequisites

Completion of all previous lab guides (up to Lab Guide-09) is required before proceeding with Lab Guide-10.

Software Requirement

- **Docker Desktop**: Installed on your Windows machine.
- **Docker Compose**: If using multi-container setups (optional).
- **TodoAPP_MYSQl**: To download the source folder **click here**

Hardware Requirement

- Minimum of 4 GB RAM

- At least 2 cores in the processor

Implementation Steps

Step-1: Restrict Docker Container Capabilities

By default, Docker containers have a range of Linux kernel capabilities that are not always necessary. Reducing these capabilities will limit what the container can do in case of a security breach.

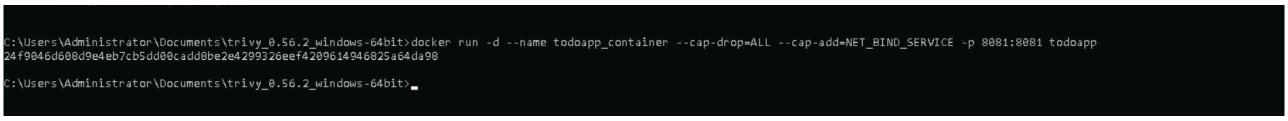
1.1 Dropping Unnecessary Capabilities

You can restrict what a container is allowed to do by using the `--cap-drop` option in your `docker run` command. Here is an example for the **Java TodoApp**:

```
docker run -d --name todoapp_container --cap-drop=ALL --cap-add=NET_BIND_SERVICE -p 8081:8081 todoapp
```

In this example:

- **`--cap-drop=ALL`**: Drops all default Linux capabilities from the container.
- **`--cap-add=NET_BIND_SERVICE`**: Adds only the capability to bind network services (like a web server).



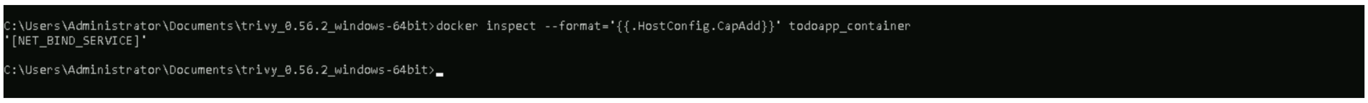
```
C:\Users\Administrator\Documents\trivy_0.56.2_windows-64bit>docker run -d --name todoapp_container --cap-drop=ALL --cap-add=NET_BIND_SERVICE -p 8081:8081 todoapp
24f9046d608d9e4eb7cb5dd00cadd8be2e4299326eef4209614946825a64da98

C:\Users\Administrator\Documents\trivy_0.56.2_windows-64bit>_
```

1.2 Verifying Container Capabilities

To verify the capabilities of a running container, you can use:

```
docker inspect --format='{{.HostConfig.CapAdd}}' todoapp_container
```

This command will list the capabilities the container has. You should see only the capabilities you explicitly added (like `NET_BIND_SERVICE`).



```
C:\Users\Administrator\Documents\trivy_0.56.2_windows-64bit>docker inspect --format='{{.HostConfig.CapAdd}}' todoapp_container
'[NET_BIND_SERVICE]'

C:\Users\Administrator\Documents\trivy_0.56.2_windows-64bit>_
```

1.3 Minimal Capabilities for Common Use Cases

- **Networking apps**: You may only need the capability to bind services (`NET_BIND_SERVICE`).
- **Non-networking apps**: It may be possible to drop **all** capabilities.

Step-2: Configure Docker User Namespaces

> Note - User namespaces currently only support **Linux containers**. This feature is not available for Windows containers on Docker Desktop.

By default, Docker containers run as the root user inside the container, which can be dangerous if exploited. To isolate container users from host users, you can configure **user namespaces**, which map container users to non-root users on the host system.

2.1 Enable User Namespaces in Docker

To enable user namespaces in Docker, follow these steps:

1. **Open Docker's daemon configuration file** (on Windows, this is typically inside Docker Desktop settings or at `C:\ProgramData\Docker\config\daemon.json`).
2. **Modify the configuration** to enable user namespaces:

```
{
  "userns-remap": "default"
}
```

This configuration remaps container users to a non-root user on the host.

2.2 Verify User Namespace Mapping

After restarting Docker, run a container and verify that user namespaces are active:

```
docker run -d --name todoapp_container my_todoapp
docker inspect todoapp_container | grep User
```

You should see that the container's user ID has been mapped to a non-root user on the host (e.g., `HostConfig.UsernsMode`).

2.3 Create a Custom User Namespace Mapping

To create custom mappings, you can define a specific user or group mapping for your containers. For example, map the container’s root user (`uid 0`) to a non-root user (`uid 1001`) on the host.

1. **Create a mapping file** on your system (e.g., `/etc/subuid` and `/etc/subgid`):

```
yourusername:100000:65536
```

This maps the container's user range starting from `100000` for 65536 users on the host.

2. **Update Docker’s `daemon.json`** with the custom mapping:

```
```

```
{
  "userns-remap": "yourusername"
}
```

3. **Restart Docker**:

```
docker-compose down
docker-compose up
```

References

- Docker Documentation: https://docs.docker.com/
- Restricting Container Capabilities: https://docs.docker.com/engine/reference/run/#runtime-privilege-and-linux-capabilities
- User Namespaces in Docker: https://docs.docker.com/engine/security/userns-remap/