Managing Docker Containers and Linux Capabilities

Behruz Sayyadi

August 18, 2024

1 Introduction

This report details the process of setting up a Docker container with specific configurations and managing file capabilities on a Linux system. The goal is to provide a practical understanding of Docker containerization and Linux capabilities, including their relevance in security and system management.

2 Linux Capabilities

2.1 Capability Sets

In Linux, capabilities are managed in different sets for a process:

- **Effective Set (e)**: This set includes capabilities that are currently active and usable by the process. If a capability is in this set, the process can use it immediately.
- **Permitted Set (p)**: This set defines the capabilities that a process is allowed to use. If a capability is in the permitted set, the process can request it if it is available in the effective set.
- **Inheritable Set (i)**: This set determines which capabilities will be inherited by child processes when the process creates them.
- **Ambient Set (a)**: This set allows a process to retain certain capabilities even if they are not in the effective set. It's useful for scenarios where a process might drop some capabilities but retain others for specific operations.

2.2 Role of the Capability Flags

Here's a detailed explanation of the flags used with the `setcap` command:

- **Permitted ('p')**:
  - The 'p' flag adds the capability to the permitted set.
  - When you specify `cap_setuid+p` with the `setcap` command, you are adding the `cap_setuid` capability to the permitted set of the file. This means:
    * The process that executes this file can request to use the `cap_setuid` capability.

    * For the process to actually use this capability, it must also be present in the effective set (`e`).
    * If a process does not have a capability in its effective set, it cannot use it, even if it is in the permitted set.

- **Effective (‘e‘)**:
  - The ‘e‘ flag sets the capability to the effective set.
  - When a capability is in the effective set, the process can use that capability immediately.
  - For example, `cap_setuid+e` makes the `cap_setuid` capability immediately available for use by the process.
- **Inheritable (‘i‘)**:
  - The ‘i‘ flag adds the capability to the inheritable set.
  - This means that child processes created by the process will inherit this capability, if the child process is allowed to use it.
  - For example, `cap_setuid+i` ensures that any child processes will inherit the `cap_setuid` capability.
- **Ambient (‘a‘)**:
  - The ‘a‘ flag adds the capability to the ambient set.
  - This set allows the process to retain certain capabilities even if they are not in the effective set.
  - For example, `cap_setuid+a` ensures that the `cap_setuid` capability is available to the process even if it is dropped from the effective set.

2.3 How Capability Flags Work

Here’s how these flags work in practice:

- **Setting the Capability**: When you use a command like `setcap cap_setuid+ep /path/to/binary`, you are:
  - Adding `cap_setuid` to the permitted set (`p`).
  - Making `cap_setuid` effective (`e`), meaning it can be used when the binary runs.
- **Permissions Check**:
  - The process must have the capability in its effective set to use it.
  - If the capability is only in the permitted set but not in the effective set, the process cannot use it.
  - If the capability is in the inheritable set, it will be passed to child processes, provided the capability is also present in the effective set.
  - If the capability is in the ambient set, the process retains it even if it is dropped from the effective set.

3 Managing Linux Capabilities

3.1 Understanding Capabilities

Linux capabilities are a set of privileges that can be assigned to processes, allowing fine-grained control over what actions a process can perform. Capabilities are part of the Linux security model that provides a way to split the privileges of the root user into distinct units.

3.2 Setting and Removing Capabilities

Capabilities can be set and removed using the `setcap` command. For example, to set `cap_setuid` on the `vim` binary:

```
sudo setcap cap_setuid+ep /usr/bin/vim.basic
```

To remove capabilities:

```
sudo setcap -r /usr/bin/vim.basic
```

3.3 Checking Capabilities

Capabilities can be verified using the `getcap` command:

```
getcap /usr/bin/vim.basic
```

If no capabilities are set, `getcap` will produce no output.

4 Docker Setup

4.1 Docker Installation

Docker was installed on a system running Ubuntu 16.04. The installation steps involved updating the package lists and installing Docker along with necessary dependencies.

```
sudo apt-get update
sudo apt-get install -y docker.io
sudo systemctl start docker
sudo systemctl enable docker
```

4.2 Creating a Dockerfile

A Dockerfile was created to build a custom container with specific configurations. The Dockerfile included instructions to install essential packages, set up users, and manage file permissions.

```
# Use Ubuntu 16.04 as the base image
FROM ubuntu:16.04

# Set environment variables to avoid interactive prompts during
   package installation
```

```
ENV DEBIAN_FRONTEND=noninteractive

# Update package lists and install necessary tools including
   getcap
RUN apt-get update && apt-get install -y \
    curl \
    vim \
    git \
    sudo \
    tree \
    less \
    libcap2-bin

# Create the user with a home directory and set passwords
RUN useradd -m -s /bin/bash minihacker && \
    echo "minihacker:password" | chpasswd && \
    echo "root[redacted] | chpasswd

# Set up permissions for directories and files
RUN mkdir -p /home/minihacker && \
    chmod 700 /home/minihacker && \
    chown minihacker:minihacker /home/minihacker

# Set capabilities on the actual vim binary
RUN ln -s /usr/bin/vim.basic /usr/bin/vim && \
    setcap cap_setuid+ep /usr/bin/vim.basic

# Set the working directory
WORKDIR /home/minihacker

# Switch to the non-root user
USER minihacker

# Default command to run when the container starts
CMD ["bash"]
```

4.3 Building and Running the Docker Container

To build the Docker image and run a container, the following commands were used:

```
docker build -t my_custom_container .
docker run -it my_custom_container
```

5 Summary and Best Practices

This report outlined the process of setting up a Docker container, managing file capabilities, and understanding their implications. Proper management of capabilities is crucial for enhancing system security and ensuring that processes have only the necessary privileges.

6 Further Reading

- Docker Documentation
- Linux Capabilities Manual