MCS: Secure Execution Environments

# Practical Exercises:<br>Linux Containers (LXC)

March 24, 2024 Due date: no date

## Changelog

- v1.0 - Initial version.

## 1 Introduction

The goal of this laboratory guide is to show how to make use of Linux Containers (LXC) and to observe how they use the protection mechanisms available in the Linux kernel.

These exercises must be executed in Linux systems, which can run on virtual machines.

## 2 Installation

Start by installing the `lxc` with your usual Linux package installer, such as `apt`:

```
apt-get install lxc
```

Run the command

```
lxc-checkconfig
```

to see how is your system prepared to run LXC containers.

## 3 Create an unprivileged container

### 3.1 Global setup

An unprivileged container runs a set of processes that have privileges within the container, while being unprivileged outside. Containers use Linux namespaces, and in this case the `user` namespace, to create a mapping between the UIDs/GIDs usable within a container and the same identifiers of the container's process outside. Usually, UIDs/GIDs are shifted upwards, meaning the 0 (super-user) inside the container corresponds to 100 000 outside, for instance. Thus, if any process with the container is able to escape its protections and access an outside resource, it will have no special privileges.

Since several users should be able to create and run LXC containers in the same shared Linux host, it is advised to create different UID/GID mappings for each user. The default, per-user, UID/GID mappings are provided by the file `/etc/subuid`. Print the contents of this file

```
cat /etc/subuid
```

and check that, for your user, you should have a line starting with your username, followed by two numbers, all separated by colons, possibly looking as follows:

```
your-username:100000:65536
```

This means that for the containers launched by `your-username`, the UIDs in the container will correspond to a value 100 000 higher (thus, 0 in the container $\rightarrow$ 100 000 outside) for UIDs for a total of 65 536 possible values (from 0 until 65 535).

The file `/etc/subgid` has the same goal and syntax, but for GIDs.

These UIDs and GIDs are called "subordinated UIDS / GIDs".

Usually it is interesting to have containers with a network interface, as it allows them to communicate with the outside network for a variety of purposes. LXC uses a bridge to do so: `lxcbr0`. You can observe the actual configuration of this interface with this command:

```
ifconfig lxcbr0
```

The interface should be up and properly configured with a private IP address, so you should see something like this:

```
lxcbr0:  flags=4099<UP,BROADCAST,MULTICAST> mtu 1500
         inet 10.0.3.1 netmask 255.255.255.0 broadcast 10.0.3.255
         ether 00:16:3e:00:00:00 txqueuelen 1000 (Ethernet)
         RX packets 0 bytes 0 (0.0 B)
         RX errors 0 dropped 0 overruns 0 frame 0
         TX packets 0 bytes 0 (0.0 B)
         TX errors 0 dropped 0 overruns 0 carrier 0 collisions 0
```

Since ordinary users do not have the power to manage bridges, you should instruct LXC whether a particular user can add new `veth` interfaces (connected to `veth` interfaces on LXC containers) to the bridge. This information is provided by means of the `/etc/lxc/lxc-usernet` file.

Edit this file (as super-user) and add to it a line with the following contents:

```
your-username veth lxcbr0 10
```

This line allows you to create and add at most 10 `veth` interfaces to the LXC bridge `lxcbr0`.

## 3.2 Container setup

To create a particular container we use a configuration file. By default, the `lxc-create` command uses the configuration file found at `~/.config/lxc/default.conf`.

Create this file from a template:

```
mkdir ~/.config/lxc
cp /etc/lxc/default.conf ~/.config/lxc/default.conf
```

and edit to append the following two lines:

```
lxc.idmap = u 0 100000 65536
lxc.idmap = g 0 100000 65536
```

This lines define the translation of mapping of UIDs and GIDs from a container to the outside system, and must match the specification for your username in `/etc/subuid` and `/etc/subgid`.

Now, you can create your container. The container will be build from a template that specifies how to create a particular distribution to run over an existing kernel. You can get a list of possible templates from the LXC maintainers. Thus, run

```
lxc-create -t download -n my-container
```

This command makes use of GPG asymmetric keys, downloaded from a central key repository, to check signatures on templates. In the case you get an error downloading those keys, you can skip the signature verification by using instead this command:

```
lxc-create -t download -n my-container -- --no-validate
```

This command will show you a list of possible distributions, releases and architectures that you can use. You can choose whatever you like, but most probably you are running over a `x86` or `x86_64` architecture, thus it does not make much sense to this experiment to choose an architecture other than `i386` or `amd64`.

Choose one suitable combination; in this guide we will assume ubuntu, bionic and amd64.

You can see the container created with the comamnd that lists your containers:

```
lxc-ls
```

or, with more (fancy) details,

```
lxc-ls -f
```

You container will be created in the directory `~/.local/share/lxc/my-container`. List the files on this directory:

```
ls -l ~/.local/share/lxc/my-container
```

You can see there a configuration file (`config`), owned by you, and a directory (`rootfs`), owned by someone that has no name (it should be 100 000 for both the UID and GID of the owner). These are the identifiers of root (UID and GID 0) processes running on the container. This directory is the root of the file system visible inside the container, and it is owned by root. But the containers root processes, outside, for this container, are given a UID and a GID of 100 000. Thus, this allows root processes within the container (with UID 0) to access files of its file system that are stored outside the container (with UID 100 000). Confusing? Maybe a bit...

Just to have a notion of the space occupied by your container, execute:

```
du -sh ~/.local/share/lxc/my-container/rootfs
```

You may see some errors, because you cannot enter into some diretories (they are not owned by you...), but they are not likely to change the outcome, which should be around 0.5 GiB.

Observe the contents of the container configuration file:

```
cat ~/.local/share/lxc/my-container/config
```

You can observe a series of instructions that LXC should follow to create this container. You have instructions for the distribution you have chosen, you have container specifix configurations (UID/GID mappings, the path of the container's root directory, the container name) and its network configuration, defined as default for you (user).

## 4 Run a single command in the container

You can use a container to run a single command, which can update part of the container state or not. For that, you can use the `lxc-execute` command, as in this example:

```
lxc-execute -n my-container -- ls -la
```

Note: `--` is the "standard" way of expressing "no more command options after this mark".

You can use this comand to set a user password. As we chose a Ubuntu distribution, the username will be `ubuntu`, so you can execute:

```
lxc-execute -n my-container -- cat /etc/shadow
lxc-execute -n my-container -- passwd ubuntu
lxc-execute -n my-container -- cat /etc/shadow
```

You should observe the addition of the transformed password to the `ubuntu` account.

## 5 Run a container for more than one command

You can run a container in foreground or background. In foreground, it will use your console as its, and you will be presented with an ordinary console login. In background, the container has no visible interface, and you will interact with it through its LXC-specific interface.

### 5.1 Foreground

To run the container we have created in foreground execute the following command:

```
lxc-start -n my-container -F
```

You will see something that you ordinarily observe during a Linux boot, and finally a console login is presented to you. Since you have just set up the password for user `ubuntu`, you can use it an log in.

Upon logging in, check your identification inside the container:

```
id
```

As you can see, you are also a sudoer (you belong to the `sudo` group).

From here, you can execute all kinds of commands available inside the container. For instance, you should have Internet access (if the container's host has, necessarily), so you can ping some host:

```
ping 1.1.1.1
```

Shutdown the container the same way you do it in a Linux system:

```
sudo shutdown -h now
```

### 5.2 Background

To run the container we have created in background execute the following command:

```
lxc-start -n my-container -d
```

You can confirm that the container is running by running:

```
lxc-ls -f
```

and observing the information provided.

Let's inspect some external characteristics about the container. Execute the following command to observe some run-time details.

```
lxc-info -n my-container
```

This command shows the PID of the first process of the container (internal PID equal to 1). Using this PID, you can see the file that contains the executable running on this process:

```
cat /proc/PID/cmdline
```

where PID is the above referred PID. You should see `/sbin/init`, which is the initial application that is run in a Linux operating system.

You can also see the IP address of the container, which you can ping.

### 5.3 Using a background container

To initiate a login in a background container, you can use the following command:

```
lxc-console -n my-container
```

You will see a login console just like you have in Linux terminal consoles.

You can login, and then logout, and you return to the login dialog. To exit this application, follow the instructions provided (type the sequence `CTRL+a q`).

Now, to “enter” into a root console within the container, you can execute:

```
lxc-attach -n my-container
```

and there you are. Now you can run anything as root inside the container.

Exit this console and the application terminates.

## 6 Freezing and stopping a container

Frezzing a container means stopping all of its processes. This is similar to the hybernation of an operating system. Execute:

```
lxc-freeze -n my-container
```

Check the status of the container:

```
lxc-ls -f
```

For unfreezing it, use:

```
lxc-unfreeze -n my-container
```

Check the status of the container:

```
lxc-ls -f
```

Finally, you can stop a running container with this command:

```
lxc-stop -n my-container
```