Introduction and Goal

Linux containers, called LXCs, provide a lightweight virtualization and independent way to host isolated applications. Linux containers are used to host the application and maintain its dependency within a separate container or entity and is resource efficient. Furthermore, there are various advantages of using a Linux container in enterprise businesses; therefore, it is gaining a lot of traction nowadays. Companies are working to understand the different ways in which these Linux containers could be used because of how versatile they are.

The main goals of this lab are to comprehend the significance of Linux Containers (LXCs), their adaptability and critical concepts, setting up basic to advanced configurations, understanding operating system-level virtualization, and managing container networking. In addition, the lab also focuses on the overall security posture of Linux containers, various vulnerabilities and their mitigation, and security threats related to LXCs.

Architecture Diagram

To perform the lab, a few requirements had to be put in place to ensure the Linux containers were up and running and that each container had a dedicated static IP so that the LXCs could communicate with one another.

Different kinds of virtualization software are available today, each based on the CPU architecture of the physical host machines. For example, Macs with Apple silicon chips and ARM architecture have virtualization software like UTM and Vmware Fusion.

Secondly, once the Debian-based image is virtualized on the virtualization software, the Linux Containers have to be created. Each container also needed a static IP address, and the containers were set up so that each LXC could ping the other ones created.

Finally, the LXCs (A, B and C) should be up and running, with specified static IP addresses on the virtualized Debian image.

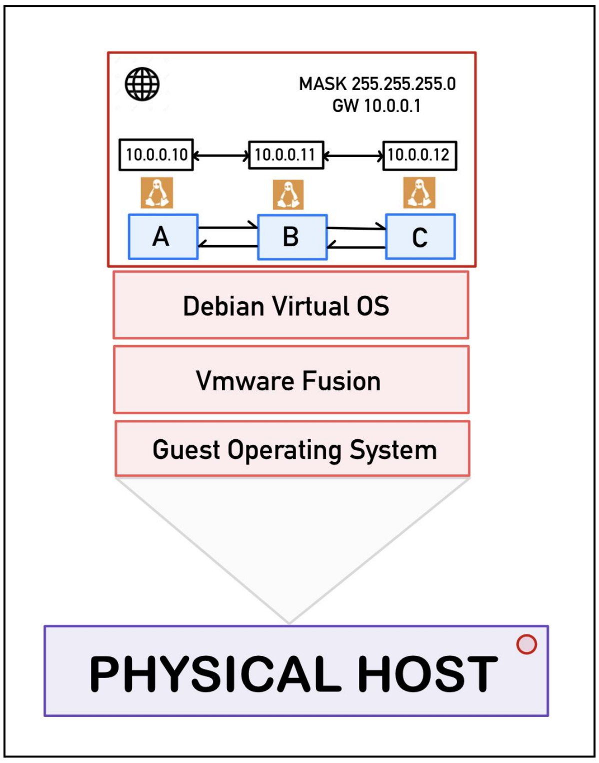


Figure 1. Architecture Diagram of the Setup

Procedure and Commands Used

To begin, a debian based image with GUI capabilities is virtualized on the vmware software on the physical host machine. After the initial setup and configuration, the below commands are run in the terminal in the same sequence as mentioned below.

1. **Setting up SSH to initiate SSH connection from host machine to the virtualized operating system**
   ```
   sudo apt-get install openssh-server
   ```

2. **To check the SSH status on the virtualized operating system**
   ```
   sudo systemctl status sshd
   ```

3. **Installing LXC package via terminal**
   ```
   sudo apt-get update && sudo apt-get install lxc
   ```

4. **Installing additional packages for LXC**
   ```
   sudo apt install lxc-templates
   ```

5. **Installing Bridge Utils software**
   ```
   sudo apt install bridge-utils
   ```

6. **Setting up the network interface**
   ```
   sudo brctl addbr virbr0
   ```

7. **Creating the containers one-by-one**
   ```
   sudo lxc-create --name A --template ubuntu
   sudo lxc-create --name B --template ubuntu
   sudo lxc-create --name C --template ubuntu
   ```

8. **Allocating specific memory to the containers**
   ```
   sudo lxc-cgroup -n A memory.limit_in_bytes 256000000
   sudo lxc-cgroup -n B memory.limit_in_bytes 256000000
   sudo lxc-cgroup -n C memory.limit_in_bytes 256000000
   ```

9. **Limiting the CPU cores for specific container**
   ```
   sudo lxc-cgroup -n A cpuset.cpus "0,1"
   sudo lxc-cgroup -n B cpuset.cpus "0,1"
   sudo lxc-cgroup -n C cpuset.cpus "0,1"
   ```

10. **Listing the installed containers via terminal**
    ```
    sudo lxc-ls --fancy
    ```

11. **Starting the containers A, B and C**
    ```
    sudo lxc-start -n A
    sudo lxc-start -n B
    sudo lxc-start -n C
    ```

12. **Logging into the container with console access. The default username and password is always set to “ubuntu”**
    ```
    sudo lxc-console --name A
    sudo lxc-console --name B
    sudo lxc-console --name C
    ```

13. **Checking the debian release of each container**
    ```
    lsb_release -a
    ```

14. **Installing net tools individually on each LXC (A, B and C) to simplify the networking tools visibility**
    ```
    sudo apt-get install -y net-tools
    ```

15. **To check the gateway**
    ```
    netstat -rn
    ```

16. **updating and patching each container**
    ```
    sudo apt-get update
    sudo apt-get upgrade
    ```

17. **Installing nano to edit and modify the networking or any configuration files**
    ```
    sudo apt install nano
    ```

18. **Configuring the static IP addresses on each container individually**
    sudo nano /etc/netplan/10-lxc.yaml

19. **Actual and unchanged default “10-lxc.yaml” contents are specified as**

```
network:
  version: 2
  ethernets:
    eth0: {dhcp4: true}
```

To change the IP address from dynamic assignment to static assignment, the below content must be pasted on the 10-lxc.yaml file

Container A
```
network:
    version: 2
    renderer: networkd
    ethernets:
        eth0:
            addresses:
                - 10.0.0.10/24
            routes:
                - to: default
                  via: 10.0.0.1
```

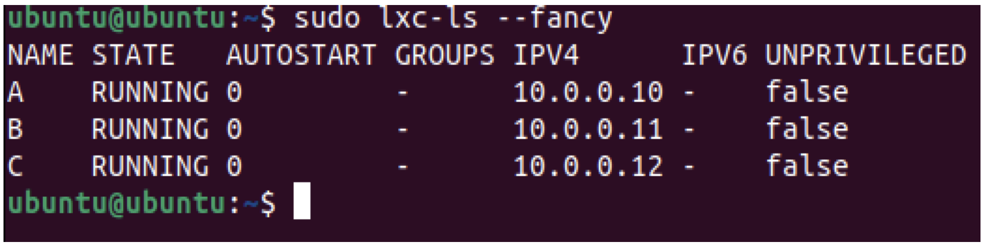
Container B
```
network:
    version: 2
    renderer: networkd
    ethernets:
        eth0:
            addresses:
            - 10.0.0.11/24
            routes:
                - to: default
                  via:10.0.0.1
```

Container C
```
network:
    version: 2
    renderer: networkd
    ethernets:
        eth0:
            addresses:
                -10.0.0.12/24
            routes:
                - to: default
                  via:10.0.0.1
```

20. **To apply the changes that were made on “10-lxc.yaml” file**
    sudo netplan apply

LXC List and Network Configuration

Finally, after all the configuration changes are done, the LXCs should be successfully created with specified static IP addresses. The PING from A - B, B - C and A - C is verified by the command **“Ping 10.0.0.X”** from each container.



```
ubuntu@ubuntu:~$ sudo lxc-ls --fancy
NAME STATE   AUTOSTART GROUPS IPV4      IPV6 UNPRIVILEGED
A    RUNNING 0         -      10.0.0.10 -    false
B    RUNNING 0         -      10.0.0.11 -    false
C    RUNNING 0         -      10.0.0.12 -    false
ubuntu@ubuntu:~$
```

Figure. 2. List of created containers with specified static IP addresses

Question 1

Linux containers are used extensively in production environments to host separate services and applications. Compared to traditional virtual machines, these Linux Containers have a lot of advantages. Most of the time, the LXCs share the host machine's kernel. This makes the environment for each service and application efficient and lightweight. Also, when the LXCs are independent, it gives the hosted application the security and isolation it needs.

The LXCs also tend to use few resources, which makes them more efficient. The containers can also be started and stopped quickly, which makes them ideal for developing and deploying production applications with minimal downtime. Most businesses want to have as little downtime and change windows as possible. With containers, downgrading or rolling back the changes pushed to the production containers is instant, making it an ideal environment to host all the critical production applications. [1]

Question 2

Linux Containers and Virtual Machines differ in terms of usability, efficiency and approach in managing and hosting the production applications. Usually, the virtual machines are complete systems that run on the host's operating system. These virtual machines include the operating system and hardware required on top of the Host OS.
Furthermore, each VM that runs within the host OS is isolated from each other, providing excellent security, and different VMs can host various operating systems and can hold different applications. However, these virtual machines use a lot of resources because each operating system in the VM requires its library and boot files to perform as an independent Virtual Machine.

Whereas the Linux Containers (LXCs) use a shared operating system kernel. These Linux Containers provide excellent isolation and offer security but share the kernel and all the libraries required with the host OS. Therefore, due to the usage of shared kernels and libraries, the LXCs are lightweight and highly efficient. Likewise, resource utilization is very minimal, and due to this behavior, it is possible to host several containers on the same host operating system within a single physical machine.

When it comes to efficiency, Linux Containers are more efficient than traditional virtual machines. Because LXCs use few resources and share the kernel, they are very lightweight. This helps businesses deploy changes and rollbacks in a production environment with little downtime, and it is quick to start and stop the whole container independently.

Both virtual machines and LXCs are excellent ways to host applications in isolated environments. Virtual machines are more flexible because they can simulate the whole computer system using distinct libraries. LXCs, on the other hand, could share the kernel and offer lightweight functionality, which increases efficiency. [2]

Question 3

Linux Containers (LXCs) are very useful and work well, but they also have drawbacks, restrictions, and limits.

First, LXC is known for using as few resources as possible from the physical host machine. However, sometimes it uses all the resources from the physical host machine, slowing down other containers.

Secondly, access is usually limited when it comes to accessing the containers. This is because the containers are isolated, making administering the containerized applications uneasy. Moreover, since LXC uses a shared kernel, the kernel version plays a significant role and therefore, different kernel versions cause incompatibility issues when migrating the containers.

Also, when it comes to security, LXCs are usually secure and isolated because they use the kernel and libraries from the host physical machine and the host operating system. But, a simple flaw or vulnerability on the kernel level can cause considerable risk to the containers sharing the kernel.

Finally, Network administration is not easy with multiple LXCs and has a few limitations. It is challenging to address firewall capabilities with various containers and to administer and manage the containers; a certain level of technical expertise is required, which is considered a limitation. [3]

Question 4

The Linux Containers (LXCs) depend on a few crucial elements to provide isolation, security, and minimal resource management. Some of the most important components are explained below:

Container Runtime: This component is crucial because it manages and creates containers on the host system. The container runtime is in charge of communicating with the kernel namespaces to handle the required isolation and management of resources.

Namespaces: The kernel namespaces offer isolation to the containers at the kernel level. Furthermore, this allows each container on the host machine to have a unique view and allotment of system libraries, file system and network resources.

CGroups: These are also called "control groups," and they are a crucial part of how the containers use their resources. The CGroups provide resource management for the containers, and the administrator can view and manage the amount of resources each container is using. This is important because it ensures the container doesn't use up all the resources on the host machine.

File systems such as AUFS, a union file system, provide a layering of file systems on top of each other. This makes sure that each container has its separate file system. This also ensures that changes to the file system of one container don't hinder the file systems of other containers. [4]

Literary Review of Attacks on LXC Containers

CVE-2019-5736

It was discovered that an attacker can manipulate the process directory to run code on the host. The vulnerability was found in *runC* in 2019, but it also affected LXC.[5] The environment can be a Docker container or a privileged LXC container running on Linux, where processes have their own directories within memory mounted at */proc*. With root access in the container the attacker can overwrite the container process's binary in the */proc* directory with their own code. Starting a new container will execute the attacker's code. Because LXC and Docker are usually run with root privileges, the attacker will have then achieved root-level code execution on the host.[6][7] To prevent the attack, one should only run unprivileged containers or update their LXC version to 3.2.0 or newer.

Category: guest-to-host

CVE-2016-8649

In an unprivileged container the attacker can use an inherited file descriptor of the */proc* directory to gain access to the rest of the host's filesystem. The attacker constructs a fake */proc* inside the container and leverages it when the host next uses *lxc-attach* command. The attacker can monitor the *lxc-attach* process with the *ptrace* tool and get the host file descriptor of the entry binary. Using remote execution it can use that to access the host's filesystem.[6][8] The attack can be prevented by using LXC version 2.0.6 or newer.[9]

Category: guest-to-host

CVE-2017-18641

LXC has templates that can be run in the container. Many of the template scripts would be downloaded in plaintext over HTTP and a digital signature check is omitted. This leaves the system vulnerable to a man-in-the-middle attack as the template script can be replaced with malicious code by an attacker with access to the traffic. The vulnerability concerns only LXC 2.0, so it is advised to use another version.[10]

Category: internet-to-guest

Resource Exhaustion

A malicious party inside a container may try to exhaust the host's processor or memory resources. They can do it by running a script that keeps opening limitlessly new processes, for example, or by creating new network interfaces until the kernel runs out of memory. Countermeasures to this include setting Cgroup or user limits, which prevent the container from using more resources than allowed.[11]

Category: guest-to-host

Breaking the Isolation

1. Inside the container you can only see the container's own processes. Without exploiting vulnerabilities you cannot access processes outside of the container.

2. You can see all processes on the host with the command ***ps -A***, which also includes processes run in containers.

3. Using a script found publicly on Github we tried to take advantage of *CVE-2019-5736* to access the host from inside container C. The script was implemented in Go language, so that had to be installed in the container. It was intended for Docker so the script had to be altered to try to make it work on LXC. Finally it did print out that it had changed the mount of */proc* but it did not seem to do anything further. Access to the host was not gained. The fault may be that the code was not altered or implemented correctly to work on LXC or that the vulnerability was patched in the used environment. Another attack tested was a resource exhaustion attack. It was done with a basic command line function that calls itself until the memory resources have been depleted. It was a successful denial-of-service attack in rendering the container unusable. When the RAM limit of 256MB was set it did not affect the host, although shutting down the jammed container was significantly slower than normal. If no memory limit was set, the attack would successfully bring down the whole host.

4. Running *tcpdump* in container C does not see network traffic destined to the other containers.

5. Containers A and B seem like different host machines on the same network from the viewpoint of C. From outside the host the containerized systems would seem like hosts in another network behind a NAT. It is presumably difficult to detect from the outside if they are running in containers.

References

[1] https://www.infoworld.com/article/3197121/3-benefits-you-didnt-expect-from-linux-containers.html Cited 20.2.2023.

[2] https://www.atlassian.com/microservices/cloud-computing/containers-vs-vms Cited 20.2.2023.

[3] https://lwn.net/Articles/588309/ Cited 20.2.2023.

[4]https://subscription.packtpub.com/book/cloud-&-networking/9781788394383/1/ch01lvl1sec10/container-components Cited 20.2.2023.

[5] https://unit42.paloaltonetworks.com/breaking-docker-via-runc-explaining-cve-2019-5736/. Cited 10.2.2023.

[6] M. Reeves, D. J. Tian, A. Bianchi and Z. B. Celik, "Towards Improving Container Security by Preventing Runtime Escapes", 2021 IEEE Secure Development Conference (SecDev), Atlanta, GA, USA, 2021.

[7] https://nvd.nist.gov/vuln/detail/CVE-2019-5736. Cited 10.2.2023.

[8] https://seclists.org/oss-sec/2016/q4/515. Cited 10.2.2023.

[9] https://nvd.nist.gov/vuln/detail/CVE-2016-8649. Cited 10.2.2023.

[10] https://www.cvedetails.com/cve/CVE-2017-18641/. Cited 17.2.2023.

[11] https://linuxcontainers.org/lxc/security/. Cited 20.2.2023.