CS-E4760: Platform Security (2023)
Exercise 1: Sandboxing

Part 1: Sandboxing in Theory

Answer the following questions:

1. Process sandboxing.

a. Operating systems sandbox applications by providing each process with an independent view of memory. If this were not the case (that is, if all applications had identical page table mappings), then how could this be exploited by a malicious application? [2 points]

By giving each process its own page table, every process can pretend that it has access to the entire address memory of the processor. It does not matter if two processes may use the same address since different page tables for each process will map it to a different frame of physical memory. Every modern OS provides each process with its own address space [1]

If all applications had identical page table mappings, the frames that manage the pages would be shared among the applications. As such, any changes made by one application will be visible to the other applications. Malicious applications can exploit this loophole in three ways: stealing data, corrupting data, and gaining unauthorized privileges. First, by sharing identical page table mappings, the malicious app can observe all current data mapping, resulting in data breaches. Secondly, the malicious app can modify the data, including the pages that contain the code of other applications, causing the benign ones to crash. Finally, the malicious program can use identical mapping to trick other programs into granting it authorized access, causing security issues in the OS systems. As a result, all OSs must use virtual memory and sandboxing to isolate and protect different processes from each other.

b. Memory is commonly mapped using a 4KiB page size. Suggest an advantage and disadvantage of using a larger page size. [2 points]

In OS, memory is the physical storage for data used by the CPU. Memory is divided into two main types: RAM (random access memory) and storage (hard drives). A frame is a fixed-sized block of memory determined by the operating system that holds a single page. A page is a fixed-sized data block (such as 4kB) for virtual memory management. When a process requests memory from the OS, the system allocates the pages to the program, and when the process does not need the memory anymore, it releases the pages back to the OS. The operating system then allocates pages to other programs as requested. Therefore, the formula is that the number of frames equals the memory size divided by the number of pages.

Advantage of larger page size: When using a larger page size, the OS can map more virtual memory in fewer page table entries, which reduces the number of lookups over the page table.

This can help to improve running performance by decreasing the hardware latency of translating a virtual page address to a physical page address.

A disadvantage of using a larger page size is that it can lead to increased memory waste, also known as internal fragmentation. With a larger page size, a process can be allocated more pages of memory than it requires, leaving the redundant memory idle. Therefore, a larger page size can cause increased memory usage and reduced system resources.

c. Pages can be mapped into the application's memory space with read, write, and execute permissions. What permissions would you give to pages containing the following, and why?
i. Application code [2 points]
ii. Shared library code [2 points]
iii. Application data [2 points]

i. Application code: Read and execute permissions should be given to pages containing application code because the application code needs to be read so that the process can execute the code. Write permission should not be given because it will inadvertently cause unexpected application behaviors and allow potential attackers to modify the application directly.

ii. Shared library code: Read and execute permissions should be given to pages containing shared library code for the same reason, except several processes can now execute the same code independently. No processes are allowed to write to the shared library code with the same argument mentioned above.

iii. Application data: Read and write permissions should be given to pages containing the application data because the processes need to read the data and modify them accordingly when the application code invokes changes on the data. However, execute permission should not be given to prevent the application from interpreting its data as the application code, which can cause ambiguous code instructions and may result in process failure or security breaches.

2. VMs vs containers

a. A hypervisor provides an environment within which operating system kernels can run, whereas an operating system kernel provides an environment within which containers can run.
Describe a situation where a virtual machine must be used instead of a container to sandbox two applications running on one machine. [2 points]

A hypervisor, also known as a virtual machine monitor or VMM, is software that creates and runs virtual machines (VMs). A hypervisor allows one host computer to support multiple guest VMs by virtually sharing its resources, such as memory and processing [2]. On the other hand, an operating system is one of the most critical components that help manage computer software and hardware resources. The kernel is a core element of the OS that converts the user query into the machine language [3].

A critical distinction between them is the capability to support different systems, such as Windows, Linux, or Mac. In a hypervisor, it can support any operating system. In contrast, an operating system kernel can only support one type of container that runs in an OS compatible with the kernel itself. So, for example, the Windows kernel can run both windows and Linux containers, while Mac can only run Mac containers. Based on this difference, a situation where a virtual machine must be used instead of a container to sandbox two applications running on one machine is **when those two applications are designed to run on different OS.**

Because containers rely on the host OS, different OS containers will not be able to run on the same OS because they would not be able to share the same kernel. On the contrary, each VM runs its own OS kernel and system resources, allowing two applications to run in isolated environments. Sharing the same kernel between the two applications is no longer necessary.

b. Which component of the system interprets the application binary code in...
i A traditional virtual machine [1 point]
ii. An application virtual machine [1 point]
iii. A container system [1 point]

i. In traditional virtualization, a hypervisor virtualizes physical hardware. The result is that each virtual machine contains a guest OS, a virtual copy of the hardware the OS requires to run, and an application with its associated libraries and dependencies. VMs with different operating systems can be run on the same physical server. [4] Therefore, in a traditional virtual machine, the hypervisor is responsible for interpreting the application binary code.

ii. Application virtual machines typically allow application bytecode to run portably on many different computer architectures and operating systems. The application is usually run on the computer using an interpreter or just-in-time compilation (JIT). An example of an application virtual machine is Java Virtual Machine (JVM), where no hardware other than the processor is virtualized. Like a real computing machine, it has an instruction set and manipulates various memory areas at run time. As a result, in an application virtual machine, the application virtual machine itself is responsible for interpreting the application binary code.

iii. It is known that containers use the host OS and system resources. The containerization library, such as Docker, provides container sandboxing and relies on the host kernel to execute the application code. Therefore, in a container system, the host OS kernel is responsible for interpreting the application binary code.

Source
[1] https://bottomupcs.com/ch06s07.html
[2]https://www.vmware.com/topics/glossary/content/hypervisor.html#:~:text=A%20hypervisor%2C%20also%20known%20as,such%20as%20memory%20and%20processing.
[3] https://byjus.com/gate/difference-between-operating-system-and-kernel/
[4] https://www.ibm.com/cloud/blog/containers-vs-vms

Part 2: Creating a virtual machine

If you already have a Linux system available with Docker installed, then feel free to skip this part.

First, install a **virtual machine manager**. You have several options here:

- VirtualBox (Windows, Linux, MacOS)
- virt-manager (Linux)

If you don't already have an opinion on this, then use VirtualBox.

Next, **download an installation image** of Ubuntu, and **create a new virtual machine**, connecting its optical drive to the image that you downloaded. Then, start the virtual machine, and complete Ubuntu's installation wizard.

Install Docker by following the instructions at https://docs.docker.com/engine/install/ubuntu/#install-using-the-repository, as well as your favourite text editor or development environment.

I have already installed Virtual Box. This is the Docker installation process in the terminal

```
springnuance@springnuance-VirtualBox:~/Desktop$ sudo apt-get install docker-ce docker-ce-cli containerd.io docker-compose-plugin
Reading package lists... Done
Building dependency tree
Reading state information... Done
The following additional packages will be installed:
  docker-ce-rootless-extras docker-scan-plugin pigz slirp4netns
Suggested packages:
  aufs-tools cgroupfs-mount | cgroup-lite
The following NEW packages will be installed:
  containerd.io docker-ce docker-ce-cli docker-ce-rootless-extras docker-compose-plugin docker-scan-plugin pigz slirp4netns
0 upgraded, 8 newly installed, 0 to remove and 149 not upgraded.
Need to get 111 MB of archives.
After this operation, 428 MB of additional disk space will be used.
Do you want to continue? [Y/n] y
Get:1 http://fi.archive.ubuntu.com/ubuntu focal/universe amd64 pigz amd64 2.4-1 [57,4 kB]
Get:2 https://download.docker.com/linux/ubuntu focal/stable amd64 containerd.io amd64 1.6.15-1 [27,7 MB]
Get:3 http://fi.archive.ubuntu.com/ubuntu focal/universe amd64 slirp4netns amd64 0.4.3-1 [74,3 kB]
Get:4 https://download.docker.com/linux/ubuntu focal/stable amd64 docker-ce-cli amd64 5:20.10.22~3-0~ubuntu-focal [41,5 MB]
Get:5 https://download.docker.com/linux/ubuntu focal/stable amd64 docker-ce amd64 5:20.10.22~3-0~ubuntu-focal [20,5 MB]
Get:6 https://download.docker.com/linux/ubuntu focal/stable amd64 docker-ce-rootless-extras amd64 5:20.10.22~3-0~ubuntu-focal [8 395 kB]
Get:7 https://download.docker.com/linux/ubuntu focal/stable amd64 docker-compose-plugin amd64 2.14.1~ubuntu-focal [9 562 kB]
Get:8 https://download.docker.com/linux/ubuntu focal/stable amd64 docker-scan-plugin amd64 0.23.0~ubuntu-focal [3 622 kB]
```

```
springnuance@springnuance-VirtualBox:~/Desktop$ sudo docker run hello-world
Unable to find image 'hello-world:latest' locally
latest: Pulling from library/hello-world
2db29710123e: Pull complete
Digest: sha256:aa0cc8055b82dc2509bed2e19b275c8f463506616377219d9642221ab53cf9fe
Status: Downloaded newer image for hello-world:latest

Hello from Docker!
This message shows that your installation appears to be working correctly.

To generate this message, Docker took the following steps:
 1. The Docker client contacted the Docker daemon.
 2. The Docker daemon pulled the "hello-world" image from the Docker Hub.
    (amd64)
 3. The Docker daemon created a new container from that image which runs the
    executable that produces the output you are currently reading.
 4. The Docker daemon streamed that output to the Docker client, which sent it
    to your terminal.

To try something more ambitious, you can run an Ubuntu container with:
 $ docker run -it ubuntu bash
```

Part 3: Creating a Docker image

Next, you will write a basic Dockerfile containing a "Hello, World" application. Start by **writing an application** in your favourite programming language that prints the line "Hello, World!". The Bash shell will be installed with Ubuntu by default, but the choice of language is not important: use whatever you prefer.

You can then start your Dockerfile with

```
FROM ubuntu:latest
```

Next, **extend the Dockerfile** to do the following:

1. Insert your application source code into the container.
2. Install any software needed to build and run your application.
3. Build your application (if necessary).
4. Copy your application to an installation directory (e.g. /usr/local/bin or /app).
5. *When the container is run*, run the application.

Now, build your image with

```
$ docker build -t app .
```

and **run it** as follows:

```
$ docker run -it app
```

Submit the Dockerfile via MyCourses [15 points].

Please check my submitted Dockerfile on MyCourse. This is the Docker building process

```
springnuance@springnuance-VirtualBox:~/Desktop/helloworld$ sudo docker build -t helloworld .
Sending build context to Docker daemon  4.608kB
Step 1/6 : FROM ubuntu:latest
 ---> 6b7dfa7e8fdb
Step 2/6 : FROM python:3.7
 ---> e497d83db7bb
Step 3/6 : COPY . /app
 ---> 95ce5764bdfc
Step 4/6 : WORKDIR /app
 ---> Running in b44380e026cd
Removing intermediate container b44380e026cd
 ---> 111b8b8c4047
Step 5/6 : RUN pip install -r requirements.txt
 ---> Running in dc7a59b9877c
Collecting numpy
  Downloading numpy-1.21.6-cp37-cp37m-manylinux_2_12_x86_64.manylinux2010_x86_64.whl (15.7 MB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 15.7/15.7 MB 7.1 MB/s eta 0:00:00
Installing collected packages: numpy
Successfully installed numpy-1.21.6
```

```
Removing intermediate container dc7a59b9877c
 ---> 0444abf161ac
Step 6/6 : CMD python helloworld.py
 ---> Running in 9ce9264af32f
Removing intermediate container 9ce9264af32f
 ---> 78c8230ebd0d
Successfully built 78c8230ebd0d
Successfully tagged helloworld:latest
springnuance@springnuance-VirtualBox:~/Desktop/helloworld$ docker run -it helloworld
Hello, world!
```

Some encountered errors and how to debug them when running docker build -t helloworld .

1. **Got permission denied while trying to connect to the Docker daemon socket at …**

This is because, to run the Docker, you need to have the root privilege. Without sudo, you can't run Docker. This is strange, because according to latest update, user should be able to run Docker without the root privileges
This fix below adds the user to the docker group
$ sudo groupadd docker
$ sudo gpasswd -a $USER docker

If all fails, just use sudo to grant all privileges
$ sudo docker build -t helloworld .
$ sudo docker run -it helloworld

2. **temporary failure in name resolution**

This is because Docker had failed to resolve DNS name. Suggested solutions at https://www.tecmint.com/resolve-temporary-failure-in-name-resolution/

First, open the resolve.conf with vim
$ sudo vim /etc/resolv.conf

Add two lines
nameserver 8.8.8.8
nameserver 8.8.4.4

Press Esc -> type :x -> Enter to save the edit

3. **Unable to prepare context: unable to evaluate symlinks in Dockerfile path**

While executing the following command sudo docker build -t helloworld .
please check that Dockerfile is present in your current working directory