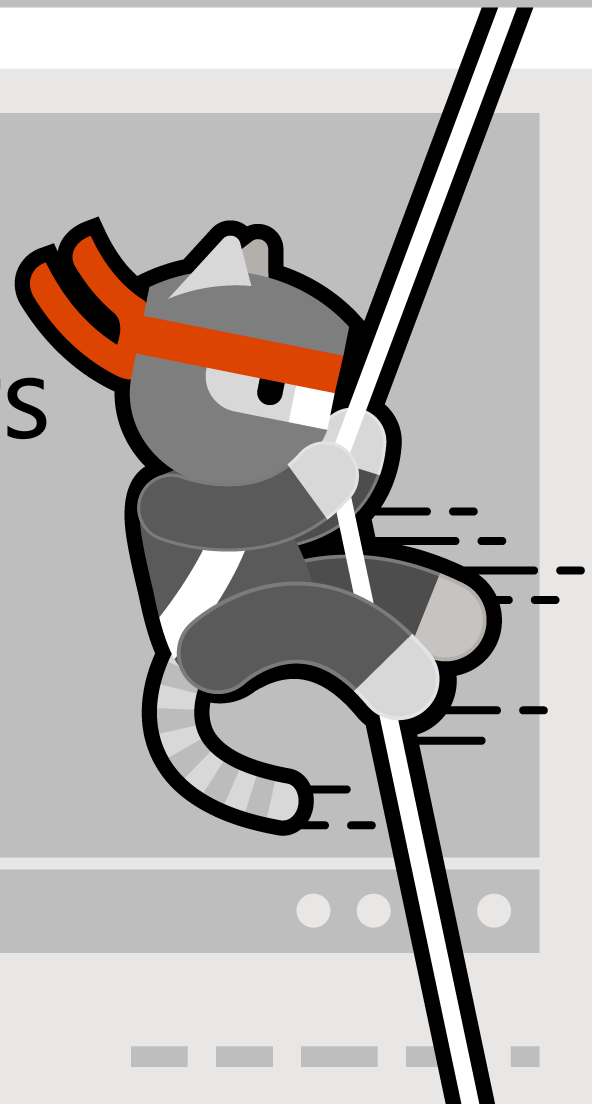
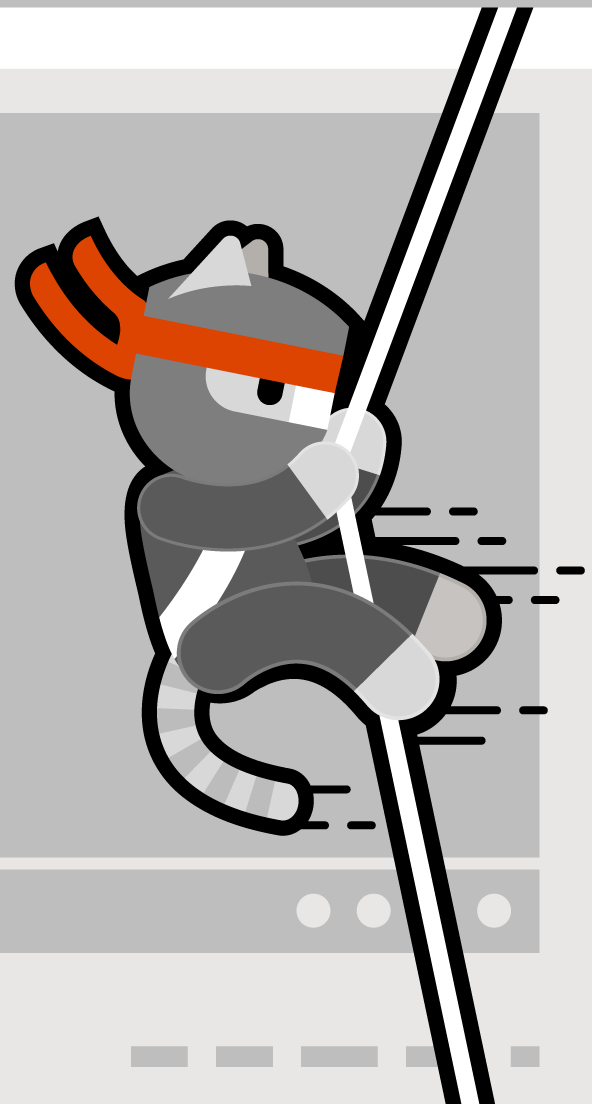
Hacking Containers

Looking at Cgroups



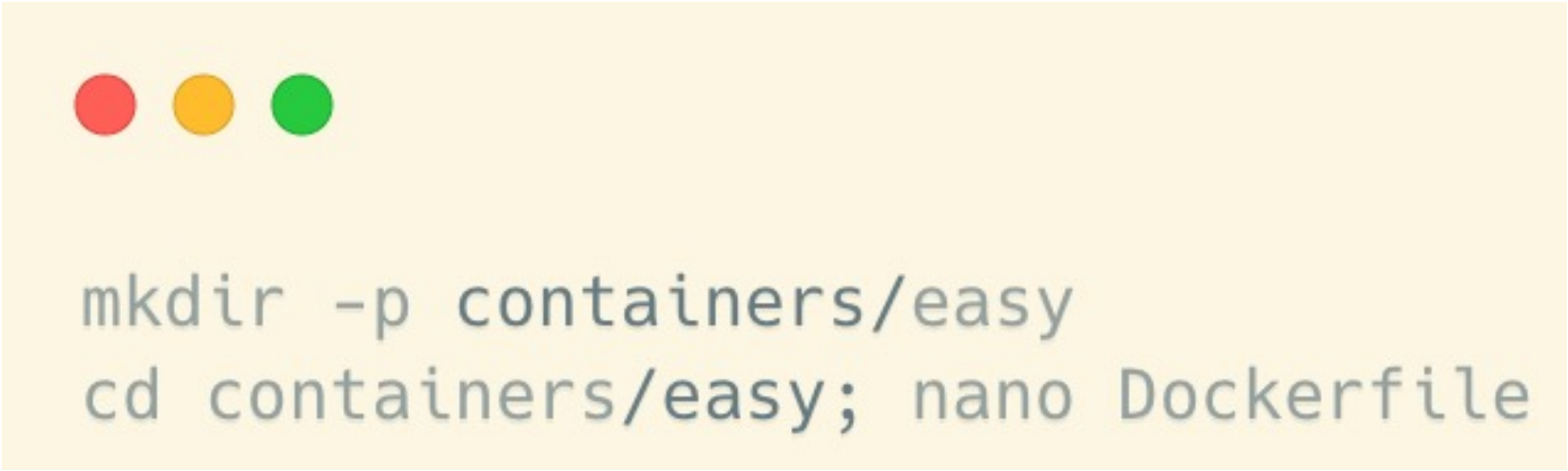
Eng Teong Cheah

Microsoft MVP



Looking at Cgroups

Inside the Kali virtual machine in the cloud, let's begin by creating a very simple container that will give us a shell:



Looking at Cgroups

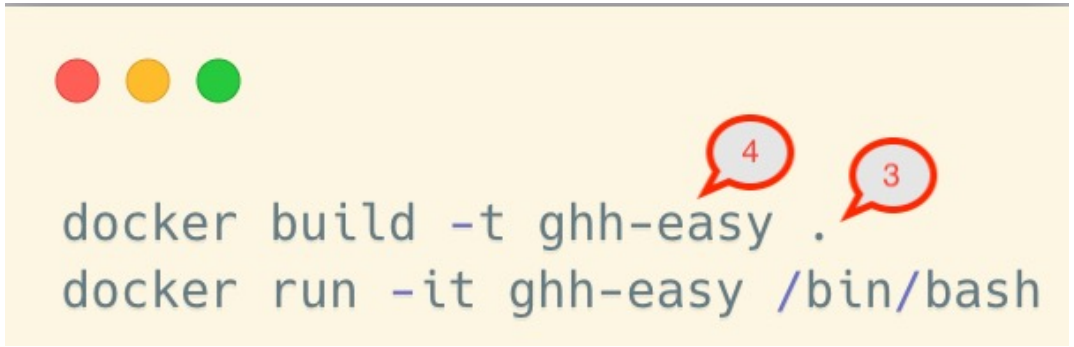
We should now be editing our Dockerfile within the containers/easy directory. The following lines should be entered into that file to be able to create a simple container:

The file we have created is known as Dockerfile.

Each and every command in the file has meaning; for example,
FROM (1) represents one command in the container and will be stored as a Singapore command in the storage file system
CMD (2) represents another command.

Looking at Cgroups

Let’s build and run our container so that we can explore cgroups:



These container commands will first build a container in the current directory (3) using the Dockerfile we created and will assign it tag of the ***ghh-easy*** (4).

We can then execute a dicker command to run the container in interactive mode.

Looking at Cgroups

The control groups on a Kali System will be based on cgroups version 2, which allows for tight controls.

One of the major differences between version 1 and version 2 is the directory hierarchy, which is viewable by using the syst file at ***/sys/fs/cgroup***.

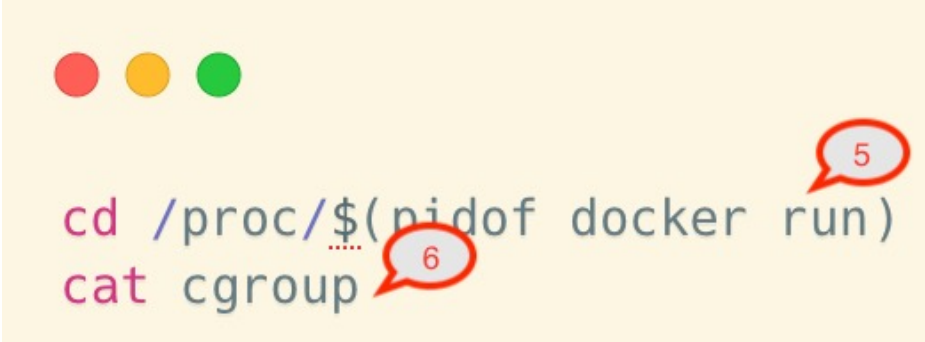
Looking at Cgroups

In cgroup version 1, each resource had its own hierarchy, and the map to namespaces:

- CPU
- cpuacct
- cpuset
- devices
- freezer
- memory
- netcls
- PIDs

Looking at Cgroups

The following commands should be performed in a new windows, as we should leave a Docker container running:



The first command will put us in the proc directory of Linux, specifically in the process ID of the running Docker container (5)

The second command will output the cgroup location that our process is running.

Looking at Cgroups

Let’s return to our Kali host. Here are some commands that can help us work with the Docker API:

docker container ls
This command shows all containers running or stopped.

docker stop
This command stops stops the containers.

docker run
This command removes a container.

Namespaces

Nsmespaces and cgroups are tightly linked, as namespaces are how the Linux Kernel can form constraints around specific items.

Namespaces, similar to how programming like C++ use them, allow for a process or collection of kernel control objects to be grouped together.

This grouping limits or controls what that process or object can see.

Namespaces

To leverage the namespace, we can use a set of APIs that are exposed by the kernel itself:

clone()
This will clone a process and then create the appropriate namespace for it.

setns()
This allows an existing process to move into a namespace that we may be able to use.

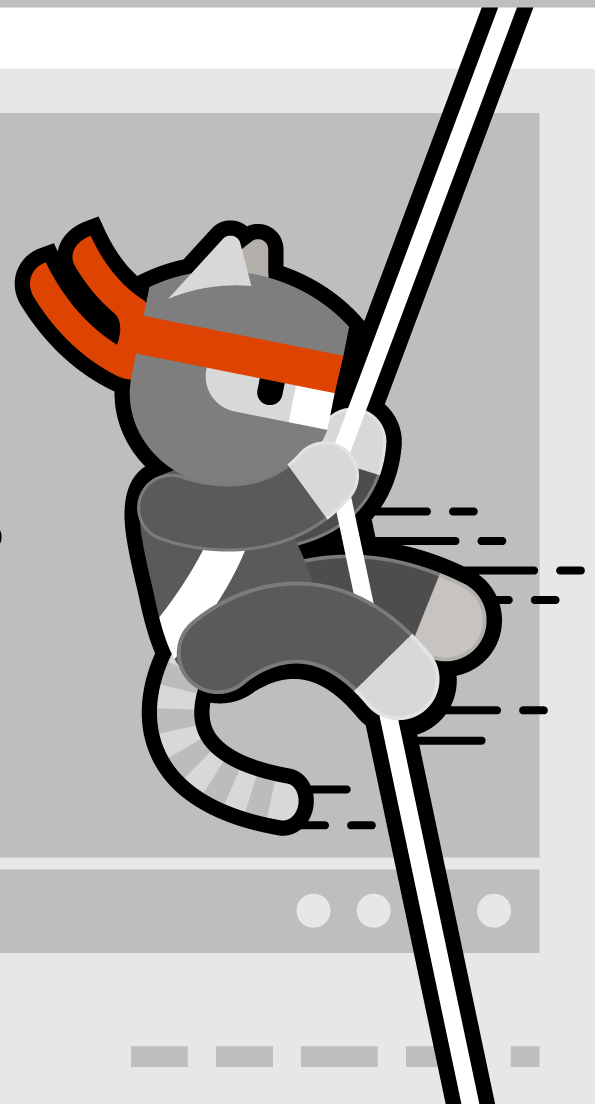
unshare()
This moves the process out of a namespace

Namespaces

You might find that exploits designed for use in the kernel outside of a container fail, and the reason they fail may have to do with the visibility the exploit has on the individual items on the disk.

You may have to rewrite your exploit to leverage a different set of APIs to move outside of a namespace and back into the global namespace.

Demo
Looking at Cgroups



References

Gray Hat Hacking, Sixth Edition