Docker In Practice

Lab – Security

For the purpose of performing permission checks, traditional UNIX implementations distinguish two categories of processes: privileged processes (whose effective user ID is 0, referred to as superuser or root), and unprivileged processes (whose effective UID is nonzero). Privileged processes bypass all kernel permission checks, while unprivileged processes are subject to full permission checking based on the process's credentials (usually: effective UID, effective GID, and supplementary group list).

Starting with kernel 2.2, Linux divides the privileges traditionally associated with superuser into distinct units, known as capabilities, which can be independently enabled and disabled. New capabilities have been added with various kernel releases. Modern kernels have over 35 different capabilities (more here: http://man7.org/linux/man-pages/man7/capabilities.7.html).

In this lab we will explore the access control features imparted by capabilities.

1. Testing Access Inside and Out

The Linux `ip` command is the modern CLI tool used to manage the IP aspects of physical and virtual devices on Linux. We can use the `ip` command to bring interfaces up and down, to name them and to assign addresses. Let's try using IP inside a container and outside of a container. Try adding a simple alias to the loopback interface on your lab host system:

```
user@ubuntu:~$ ip link set dev lo alias test77

RTNETLINK answers: Operation not permitted
```

Ah ha, normal users are not allowed to mess with the state of the network devices. RTNETLINK is a facility in the Linux operating system for user-space applications to communicate with the kernel and it is not willing to let a normal user mess up the network.

List your capabilities as a normal user:

```
user@ubuntu:~$ capsh --print

Current: =
Bounding set
=cap_chown,cap_dac_override,cap_dac_read_search,cap_fowner,cap_fsetid,cap_kill,ca
p_setgid,cap_setuid,cap_setpcap,cap_linux_immutable,cap_net_bind_service,cap_net_
broadcast,cap_net_admin,cap_net_raw,cap_ipc_lock,cap_ipc_owner,cap_sys_module,cap
_sys_rawio,cap_sys_chroot,cap_sys_ptrace,cap_sys_pacct,cap_sys_admin,cap_sys_boot
,cap_sys_nice,cap_sys_resource,cap_sys_time,cap_sys_tty_config,cap_mknod,cap_leas
e,cap_audit_write,cap_audit_control,cap_setfcap,cap_mac_override,cap_mac_admin,ca
p_syslog,cap_wake_alarm,cap_block_suspend,37
Securebits: 00/0x0/1'b0
 secure-noroot: no (unlocked)
 secure-no-suid-fixup: no (unlocked)
 secure-keep-caps: no (unlocked)
uid=1000(user)
gid=1000(user)
groups=4(adm),24(cdrom),27(sudo),30(dip),46(plugdev),110(lxd),115(lpadmin),116(sa
mbashare),999(docker),1000(user)
```

The `Current =` line tells us we have no capabilities. Normal users are only allowed to make system calls that are unrestricted by capabilities.

Display the capabilities of *root*:

```
user@ubuntu:~$ sudo capsh --print

Current: =
cap_chown,cap_dac_override,cap_dac_read_search,cap_fowner,cap_fsetid,cap_kill,cap
_setgid,cap_setuid,cap_setpcap,cap_linux_immutable,cap_net_bind_service,cap_net_b
roadcast,cap_net_admin,cap_net_raw,cap_ipc_lock,cap_ipc_owner,cap_sys_module,cap_
sys_rawio,cap_sys_chroot,cap_sys_ptrace,cap_sys_pacct,cap_sys_admin,cap_sys_boot,
cap_sys_nice,cap_sys_resource,cap_sys_time,cap_sys_tty_config,cap_mknod,cap_lease
,cap_audit_write,cap_audit_control,cap_setfcap,cap_mac_override,cap_mac_admin,cap
_syslog,cap_wake_alarm,cap_block_suspend,37+ep
Bounding set
=cap_chown,cap_dac_override,cap_dac_read_search,cap_fowner,cap_fsetid,cap_kill,ca
p_setgid,cap_setuid,cap_setpcap,cap_linux_immutable,cap_net_bind_service,cap_net_
broadcast,cap_net_admin,cap_net_raw,cap_ipc_lock,cap_ipc_owner,cap_sys_module,cap
_sys_rawio,cap_sys_chroot,cap_sys_ptrace,cap_sys_pacct,cap_sys_admin,cap_sys_boot
,cap_sys_nice,cap_sys_resource,cap_sys_time,cap_sys_tty_config,cap_mknod,cap_leas
e,cap_audit_write,cap_audit_control,cap_setfcap,cap_mac_override,cap_mac_admin,ca
p_syslog,cap_wake_alarm,cap_block_suspend,37
Securebits: 00/0x0/1'b0
 secure-noroot: no (unlocked)
 secure-no-suid-fixup: no (unlocked)
 secure-keep-caps: no (unlocked)
uid=0(root)
gid=0(root)
groups=0(root)
```

root has all capabilities in the Current section. As *root* we can do anything. Try setting the network interface *alias as *root*:

```
user@ubuntu:~$ sudo ip link set dev lo alias test77

user@ubuntu:~$ ip link ls up

1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue state UNKNOWN mode DEFAULT
group default qlen 1
    link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
    alias test77
2: ens33: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc pfifo_fast state UP
mode DEFAULT group default qlen 1000
    link/ether 00:0c:29:fe:f9:fb brd ff:ff:ff:ff:ff:ff
3: docker0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc noqueue state UP
mode DEFAULT group default
```

```
    link/ether 02:42:3a:cf:df:f7 brd ff:ff:ff:ff:ff:ff
95: veth6c26195@if94: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc noqueue
master docker0 state UP mode DEFAULT group default
    link/ether 56:e2:84:e3:de:a1 brd ff:ff:ff:ff:ff:ff link-netnsid 0

user@ubuntu:~$
```

Of course it works, *root* has all powers.

Now lets try the same experiment in a container, we'll run a container but give the container access to the host network namespace so that our container user can try to manipulate the same loopback interface we just set the alias for.

```
user@ubuntu:~$ docker container run -it --net=host ubuntu:14.04
```

While inside the container, test its namespace isolation with the following commands:

```
root@ubuntu:/# mount | grep aufs

none on / type aufs (rw,relatime,si=2843ca7f740003b8,dio,dirperm1)

root@ubuntu:/# ps -ef
UID         PID   PPID  C STIME TTY          TIME CMD
root          1      0  0 06:40 ?        00:00:00 /bin/bash
root         17      1  0 06:41 ?        00:00:00 ps -ef

root@ubuntu:/# id

uid=0(root) gid=0(root) groups=0(root)

root@ubuntu:/# ip link ls up

1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue state UNKNOWN mode DEFAULT
group default qlen 1
    link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
    alias test77
2: ens33: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc pfifo_fast state UP
mode DEFAULT group default qlen 1000
    link/ether 00:0c:29:fe:f9:fb brd ff:ff:ff:ff:ff:ff
3: docker0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc noqueue state UP
mode DEFAULT group default
    link/ether 02:42:3a:cf:df:f7 brd ff:ff:ff:ff:ff:ff
95: veth6c26195@if94: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc noqueue
master docker0 state UP mode DEFAULT group default
    link/ether 56:e2:84:e3:de:a1 brd ff:ff:ff:ff:ff:ff

root@ubuntu:/#
```

As the example commands show, we are definitely in a container. We have an AUFS root file system, the only process running is our shell and our account has changed to *root*. However, by setting the *--net=host* we have no network isolation, we have full access to the host namespace for networking.

Since we're *root* we should be able to set a network interface alias, right? Try it:

```
root@ubuntu:/# ip link set dev lo alias test-in-container

RTNETLINK answers: Operation not permitted

root@ubuntu:/#
```

Not permitted!?

- What is wrong?

Our problem is that network administration activity is not allowed within containers. Display your capabilities:

```
root@ubuntu:/# capsh --print

Current: =
cap_chown,cap_dac_override,cap_fowner,cap_fsetid,cap_kill,cap_setgid,cap_setuid,c
ap_setpcap,cap_net_bind_service,cap_net_raw,cap_sys_chroot,cap_mknod,cap_audit_wr
ite,cap_setfcap+eip
Bounding set
=cap_chown,cap_dac_override,cap_fowner,cap_fsetid,cap_kill,cap_setgid,cap_setuid,
cap_setpcap,cap_net_bind_service,cap_net_raw,cap_sys_chroot,cap_mknod,cap_audit_w
rite,cap_setfcap
Securebits: 00/0x0/1'b0
 secure-noroot: no (unlocked)
 secure-no-suid-fixup: no (unlocked)
 secure-keep-caps: no (unlocked)
uid=0(root)
gid=0(root)
groups=
```

root in a container has fewer capabilities than *root* on the host because Docker removes capabilities from the container it does not deem necessary for normal microservice style applications. We are trying to use our container as a network administration environment without CAP_NET_ADMIN. The network admin capability allows admins to perform various network-related operations:

- interface configuration
- administration of IP firewall, masquerading, and accounting
- modify routing tables
- bind to any address for transparent proxying
- set type-of-service (TOS)
- clear driver statistics
- set promiscuous mode
- enabling multicasting
- ... and setting interface aliases! (among other things)

So this container will not suit our purposes. Exit and remove the container.

2. Using Privileged Containers

The `container run --privileged` switch returns all root capabilities to the container run. This is a great switch to use when diagnosing problems but a terrible switch to use in production. We'll try it here to see if our previous failure was indeed related to lack of capability.

Run the same container as last time but add the `--privileged` switch:

```
user@ubuntu:~$ docker container run -it --net=host --privileged ubuntu:14.04

root@ubuntu:/# ip link ls up

1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue state UNKNOWN mode DEFAULT
group default qlen 1
    link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
    alias test77
2: ens33: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc pfifo_fast state UP
mode DEFAULT group default qlen 1000
    link/ether 00:0c:29:fe:f9:fb brd ff:ff:ff:ff:ff:ff
3: docker0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc noqueue state UP
mode DEFAULT group default
    link/ether 02:42:3a:cf:df:f7 brd ff:ff:ff:ff:ff:ff
95: veth6c26195@if94: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc noqueue
master docker0 state UP mode DEFAULT group default
    link/ether 56:e2:84:e3:de:a1 brd ff:ff:ff:ff:ff:ff

root@ubuntu:/#
```

Ok now lets see if we can set the alias:

```
root@ubuntu:/# ip link set dev lo alias test-in-container

root@ubuntu:/# ip link ls up

1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue state UNKNOWN mode DEFAULT
group default qlen 1
    link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
    alias test-in-container
2: ens33: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc pfifo_fast state UP
mode DEFAULT group default qlen 1000
    link/ether 00:0c:29:fe:f9:fb brd ff:ff:ff:ff:ff:ff
3: docker0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc noqueue state UP
mode DEFAULT group default
    link/ether 02:42:3a:cf:df:f7 brd ff:ff:ff:ff:ff:ff
95: veth6c26195@if94: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc noqueue
master docker0 state UP mode DEFAULT group default
    link/ether 56:e2:84:e3:de:a1 brd ff:ff:ff:ff:ff:ff

root@ubuntu:/#
```

It worked!

3. Adding Capabilities

While it is nice to have a container that can manipulate the network devices, the *--privileged* container can do much more. For example a privileged container has CAP_SYS_ADMIN, which lets it perform a wide range of admin tasks, like setting the domain name, exceeding system limits and other extreme things. Privileged containers also have CAP_SYS_BOOT, which lets you reboot the system!

Obviously it is better to provide only the permissions needed when possible. Of course using the docker run *--cap-add* switch we can add just the permission we need, CAP_NET_ADMIN.

Shutdown and delete any prior containers and try the experiment again, this time adding the net admin capability:

```
user@ubuntu:~$ docker container run -it --net=host --cap-add=NET_ADMIN
ubuntu:14.04

root@ubuntu:/# ip link ls up

1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue state UNKNOWN mode DEFAULT
group default qlen 1
    link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
    alias test-in-container
2: ens33: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc pfifo_fast state UP
mode DEFAULT group default qlen 1000
    link/ether 00:0c:29:fe:f9:fb brd ff:ff:ff:ff:ff:ff
3: docker0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc noqueue state UP
mode DEFAULT group default
    link/ether 02:42:3a:cf:df:f7 brd ff:ff:ff:ff:ff:ff
95: veth6c26195@if94: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc noqueue
master docker0 state UP mode DEFAULT group default
    link/ether 56:e2:84:e3:de:a1 brd ff:ff:ff:ff:ff:ff

root@ubuntu:/# ip link set dev lo alias test-cap-add

root@ubuntu:/# ip link ls up

1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue state UNKNOWN mode DEFAULT
group default qlen 1
    link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
    alias test-cap-add
2: ens33: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc pfifo_fast state UP
mode DEFAULT group default qlen 1000
    link/ether 00:0c:29:fe:f9:fb brd ff:ff:ff:ff:ff:ff
3: docker0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc noqueue state UP
mode DEFAULT group default
    link/ether 02:42:3a:cf:df:f7 brd ff:ff:ff:ff:ff:ff
95: veth6c26195@if94: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc noqueue
master docker0 state UP mode DEFAULT group default
    link/ether 56:e2:84:e3:de:a1 brd ff:ff:ff:ff:ff:ff

root@ubuntu:/#
```

Perfect, we now have a container with the permissions it needs but no more.

You can use the `capsh --print` command to display the capabilities available to your container:

```
root@ubuntu:/# capsh --print

Current: =
cap_chown,cap_dac_override,cap_fowner,cap_fsetid,cap_kill,cap_setgid,cap_setuid,c
ap_setpcap,cap_net_bind_service,cap_net_admin,cap_net_raw,cap_sys_chroot,cap_mkno
d,cap_audit_write,cap_setfcap+eip
Bounding set
=cap_chown,cap_dac_override,cap_fowner,cap_fsetid,cap_kill,cap_setgid,cap_setuid,
cap_setpcap,cap_net_bind_service,cap_net_admin,cap_net_raw,cap_sys_chroot,cap_mkn
od,cap_audit_write,cap_setfcap
Securebits: 00/0x0/1'b0
 secure-noroot: no (unlocked)
 secure-no-suid-fixup: no (unlocked)
 secure-keep-caps: no (unlocked)
uid=0(root)
gid=0(root)
groups=

root@ubuntu:/#
```

Our new addition, cap_net_admin, makes the necessary operations possible.

Congratulations, you have completed the Docker security lab!