Helec Bastien
Etzweiler Randall

TP R4D10 sécurisation des containers docker

3 Utilisation des namespaces par Docker

3.1 Accéder au namespace de l'hôte depuis un container Docker c'est mal.

Créez un container et vérifiez que les options suivantes de docker run "–pid=host" et "–net=host" permettent d'accéder aux processus et au réseau de l'hôte.

Création du container avec accès au namespace de l'hôte :

1. Lancement d'un container avec l'option --pid=host :

docker run -it --rm --pid=host debian:12 bash

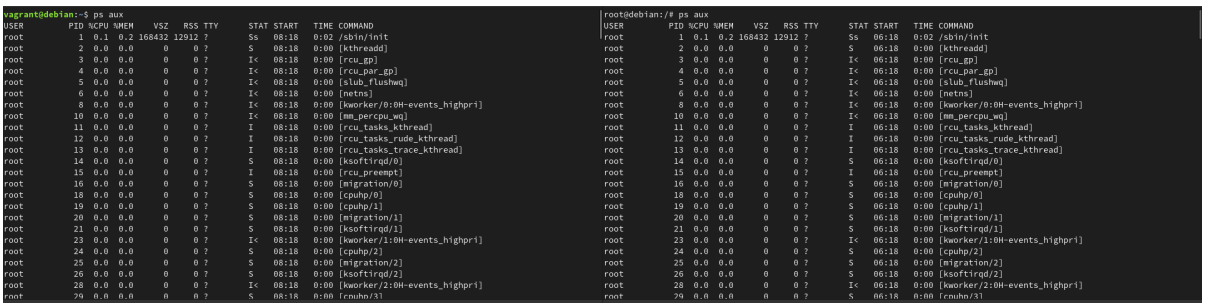
```
test@Ndeye-debian:~$ docker run -it --rm --pid=host debian:12 bash
Unable to find image 'debian:12' locally
12: Pulling from library/debian
1468e7ff95fc: Already exists
Digest: sha256:1aadfee8d292f64b045adb830f8a58bfacc15789ae5f489a0fedcd517a862cb9
Status: Downloaded newer image for debian:12
root@536e566cde84:/# ps aux
```

Une fois dans le container, on peut exécuter la commande ps aux pour voir tous les processus en cours sur la machine hôte, ce qui ne serait pas possible sans l'option --pid=host.

NB : Etant donné que la commande ps n'est pas installée par défaut dans l'image minimale de Debian que nous utilisons, pour pouvoir utiliser ps dans le container, on devra d'abord installer le paquet procps, qui contient la commande ps ainsi que d'autres outils de gestion des processus. :

apt update
apt install -y procps

Processus de l'hôte :



```
vagrant@debian:~$ ps aux
USER         PID %CPU %MEM    VSZ   RSS TTY      STAT START   TIME COMMAND
root           1  0.1  0.2 168432 12912 ?        Ss   08:18   0:02 /sbin/init
root           2  0.0  0.0      0     0 ?        S    08:18   0:00 [kthreadd]
root           3  0.0  0.0      0     0 ?        I<   08:18   0:00 [rcu_gp]
root           4  0.0  0.0      0     0 ?        I<   08:18   0:00 [rcu_par_gp]
root           5  0.0  0.0      0     0 ?        I<   08:18   0:00 [slub_flushwq]
root           6  0.0  0.0      0     0 ?        I<   08:18   0:00 [netns]
root           8  0.0  0.0      0     0 ?        I<   08:18   0:00 [kworker/0:0H-events_highpri]
root          10  0.0  0.0      0     0 ?        I<   08:18   0:00 [mm_percpu_wq]
root          11  0.0  0.0      0     0 ?        I    08:18   0:00 [rcu_tasks_kthread]
root          12  0.0  0.0      0     0 ?        I    08:18   0:00 [rcu_tasks_rude_kthread]
root          13  0.0  0.0      0     0 ?        I    08:18   0:00 [rcu_tasks_trace_kthread]
root          14  0.0  0.0      0     0 ?        S    08:18   0:00 [ksoftirqd/0]
root          15  0.0  0.0      0     0 ?        I    08:18   0:00 [rcu_preempt]
root          16  0.0  0.0      0     0 ?        S    08:18   0:00 [migration/0]
root          18  0.0  0.0      0     0 ?        S    08:18   0:00 [cpuhp/0]
root          19  0.0  0.0      0     0 ?        S    08:18   0:00 [cpuhp/1]
root          20  0.0  0.0      0     0 ?        S    08:18   0:00 [migration/1]
root          21  0.0  0.0      0     0 ?        S    08:18   0:00 [ksoftirqd/1]
root          23  0.0  0.0      0     0 ?        I<   08:18   0:00 [kworker/1:0H-events_highpri]
root          24  0.0  0.0      0     0 ?        S    08:18   0:00 [cpuhp/2]
root          25  0.0  0.0      0     0 ?        S    08:18   0:00 [migration/2]
root          26  0.0  0.0      0     0 ?        S    08:18   0:00 [ksoftirqd/2]
root          28  0.0  0.0      0     0 ?        I<   08:18   0:00 [kworker/2:0H-events_highpri]
root          29  0.0  0.0      0     0 ?        S    08:18   0:00 [cpuhp/3]
```

```
root@debian:/# ps aux
USER         PID %CPU %MEM    VSZ   RSS TTY      STAT START   TIME COMMAND
root           1  0.1  0.2 168432 12912 ?        Ss   06:18   0:02 /sbin/init
root           2  0.0  0.0      0     0 ?        S    06:18   0:00 [kthreadd]
root           3  0.0  0.0      0     0 ?        I<   06:18   0:00 [rcu_gp]
root           4  0.0  0.0      0     0 ?        I<   06:18   0:00 [rcu_par_gp]
root           5  0.0  0.0      0     0 ?        I<   06:18   0:00 [slub_flushwq]
root           6  0.0  0.0      0     0 ?        I<   06:18   0:00 [netns]
root           8  0.0  0.0      0     0 ?        I<   06:18   0:00 [kworker/0:0H-events_highpri]
root          10  0.0  0.0      0     0 ?        I<   06:18   0:00 [mm_percpu_wq]
root          11  0.0  0.0      0     0 ?        I    06:18   0:00 [rcu_tasks_kthread]
root          12  0.0  0.0      0     0 ?        I    06:18   0:00 [rcu_tasks_rude_kthread]
root          13  0.0  0.0      0     0 ?        I    06:18   0:00 [rcu_tasks_trace_kthread]
root          14  0.0  0.0      0     0 ?        S    06:18   0:00 [ksoftirqd/0]
root          15  0.0  0.0      0     0 ?        I    06:18   0:00 [rcu_preempt]
root          16  0.0  0.0      0     0 ?        S    06:18   0:00 [migration/0]
root          18  0.0  0.0      0     0 ?        S    06:18   0:00 [cpuhp/0]
root          19  0.0  0.0      0     0 ?        S    06:18   0:00 [cpuhp/1]
root          20  0.0  0.0      0     0 ?        S    06:18   0:00 [migration/1]
root          21  0.0  0.0      0     0 ?        S    06:18   0:00 [ksoftirqd/1]
root          23  0.0  0.0      0     0 ?        I<   06:18   0:00 [kworker/1:0H-events_highpri]
root          24  0.0  0.0      0     0 ?        S    06:18   0:00 [cpuhp/2]
root          25  0.0  0.0      0     0 ?        S    06:18   0:00 [migration/2]
root          26  0.0  0.0      0     0 ?        S    06:18   0:00 [ksoftirqd/2]
root          28  0.0  0.0      0     0 ?        I<   06:18   0:00 [kworker/2:0H-events_highpri]
root          29  0.0  0.0      0     0 ?        S    06:18   0:00 [cpuhp/3]
```

Container :

```
root@536e566cde84:/# ps aux
USER          PID %CPU %MEM    VSZ   RSS TTY      STAT START   TIME COMMAND
root            1  0.2  0.0 168976 13520 ?        Ss   06:34   0:01 /sbin/init
root            2  0.0  0.0      0     0 ?        S    06:34   0:00 [kthreadd]
root            3  0.0  0.0      0     0 ?        I<   06:34   0:00 [rcu_gp]
root            4  0.0  0.0      0     0 ?        I<   06:34   0:00 [rcu_par_gp]
root            5  0.0  0.0      0     0 ?        I<   06:34   0:00 [slub_flushwq]
root            6  0.0  0.0      0     0 ?        I<   06:34   0:00 [netns]
root            8  0.0  0.0      0     0 ?        I<   06:34   0:00 [kworker/0:0H-events
root            9  0.1  0.0      0     0 ?        I    06:34   0:00 [kworker/u16:0-flush
root           10  0.0  0.0      0     0 ?        I<   06:34   0:00 [mm_percpu_wq]
root           11  0.0  0.0      0     0 ?        I    06:34   0:00 [rcu_tasks_kthread]
root           12  0.0  0.0      0     0 ?        I    06:34   0:00 [rcu_tasks_rude_kthr
root           13  0.0  0.0      0     0 ?        I    06:34   0:00 [rcu_tasks_trace_kth
root           14  0.0  0.0      0     0 ?        S    06:34   0:00 [ksoftirqd/0]
root           15  0.1  0.0      0     0 ?        I    06:34   0:00 [rcu_preempt]
root           16  0.0  0.0      0     0 ?        S    06:34   0:00 [migration/0]
root           17  0.0  0.0      0     0 ?        I    06:34   0:00 [kworker/0:1-mm_perc
root           18  0.0  0.0      0     0 ?        S    06:34   0:00 [cpuhp/0]
root           19  0.0  0.0      0     0 ?        S    06:34   0:00 [cpuhp/1]
root           20  0.0  0.0      0     0 ?        S    06:34   0:00 [migration/1]
root           21  0.0  0.0      0     0 ?        S    06:34   0:00 [ksoftirqd/1]
root           22  0.0  0.0      0     0 ?        I    06:34   0:00 [kworker/1:0-mm_perc
```

2. Lancement d’un container avec l'option --net=host

docker run -it --rm --net=host debian:12 bash

```
exit
test@Ndeye-debian:~$ docker run -it --rm --net=host debian:12 bash
root@Ndeye-debian:/#
```

Cette fois on vérifie l’accès au réseau de l’hôte via netstat pour explorer l'environnement réseau de l'hôte depuis le container. On exécutera la commande netstat -tuln qui nous permettra de voir tous les ports ouverts et les services écoutant sur ces ports depuis le point de vue du container, ce qui reflète l'environnement réseau de l'hôte en raison de l'utilisation de l'option --net=host.

NB : On installera au préalable le paquet net-tools, qui contient netstat pour permettre à notre VM Debian 12 de disposer des outils netstat :

apt update
apt install net-tools -y

Container :

```
root@Ndeye-debian:/# netstat -tuln
Active Internet connections (only servers)
Proto Recv-Q Send-Q Local Address           Foreign Address         State
tcp        0      0 127.0.0.1:5000          0.0.0.0:*               LISTEN
tcp        0      0 0.0.0.0:48923           0.0.0.0:*               LISTEN
tcp        0      0 0.0.0.0:5001            0.0.0.0:*               LISTEN
tcp        0      0 127.0.0.1:6789          0.0.0.0:*               LISTEN
tcp        0      0 127.0.0.1:6791          0.0.0.0:*               LISTEN
tcp        0      0 0.0.0.0:46629           0.0.0.0:*               LISTEN
tcp        0      0 0.0.0.0:2049            0.0.0.0:*               LISTEN
tcp        0      0 0.0.0.0:60597           0.0.0.0:*               LISTEN
tcp        0      0 127.0.0.1:9000          0.0.0.0:*               LISTEN
tcp        0      0 0.0.0.0:52703           0.0.0.0:*               LISTEN
tcp        0      0 127.0.0.1:631           0.0.0.0:*               LISTEN
tcp        0      0 127.0.0.1:25            0.0.0.0:*               LISTEN
tcp        0      0 0.0.0.0:22              0.0.0.0:*               LISTEN
tcp        0      0 0.0.0.0:111             0.0.0.0:*               LISTEN
tcp        0      0 0.0.0.0:42083           0.0.0.0:*               LISTEN
tcp6       0      0 ::1:25                  :::*                    LISTEN
tcp6       0      0 :::6556                 :::*                    LISTEN
tcp6       0      0 ::1:631                 :::*                    LISTEN
tcp6       0      0 ::1:9000                :::*                    LISTEN
tcp6       0      0 :::8000                 :::*                    LISTEN
tcp6       0      0 :::5001                 :::*                    LISTEN
```

Machine hôte :

```
test@Ndeye-debian:~$ netstat -tuln
Connexions Internet actives (seulement serveurs)
Proto Recv-Q Send-Q Adresse locale          Adresse distante        Etat
tcp        0      0 127.0.0.1:5000          0.0.0.0:*               LISTEN
tcp        0      0 0.0.0.0:48923           0.0.0.0:*               LISTEN
tcp        0      0 0.0.0.0:5001            0.0.0.0:*               LISTEN
tcp        0      0 127.0.0.1:6789          0.0.0.0:*               LISTEN
tcp        0      0 127.0.0.1:6791          0.0.0.0:*               LISTEN
tcp        0      0 0.0.0.0:46629           0.0.0.0:*               LISTEN
tcp        0      0 0.0.0.0:2049            0.0.0.0:*               LISTEN
tcp        0      0 0.0.0.0:60597           0.0.0.0:*               LISTEN
tcp        0      0 127.0.0.1:9000          0.0.0.0:*               LISTEN
tcp        0      0 0.0.0.0:52703           0.0.0.0:*               LISTEN
tcp        0      0 127.0.0.1:631           0.0.0.0:*               LISTEN
tcp        0      0 127.0.0.1:25            0.0.0.0:*               LISTEN
tcp        0      0 0.0.0.0:22              0.0.0.0:*               LISTEN
tcp        0      0 0.0.0.0:111             0.0.0.0:*               LISTEN
tcp        0      0 0.0.0.0:42083           0.0.0.0:*               LISTEN
tcp6       0      0 ::1:25                  :::*                    LISTEN
tcp6       0      0 :::6556                 :::*                    LISTEN
tcp6       0      0 ::1:631                 :::*                    LISTEN
tcp6       0      0 ::1:9000                :::*                    LISTEN
tcp6       0      0 :::8000                 :::*                    LISTEN
tcp6       0      0 :::5001                 :::*                    LISTEN
```

On voit bien que les options de docker run "–pid=host" et "–net=host" ont permis à nos containers d'accéder aux processus et au réseau de l'hôte.

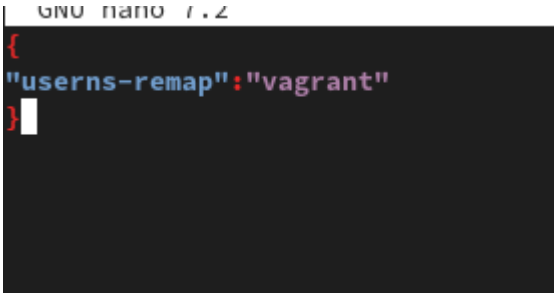
3.2 Utilisation des usernamespaces par Docker afin de limiter les droits d'un attaquant

1. Activez l'option –userns-remap avec le daemon docker afin d'activer les usernamespaces pour l'ensemble de containers.
Pour cela ajoutez dans le fichier/etc/docker/daemon.json

```
{
“user-remap” : “[user]”
}
```

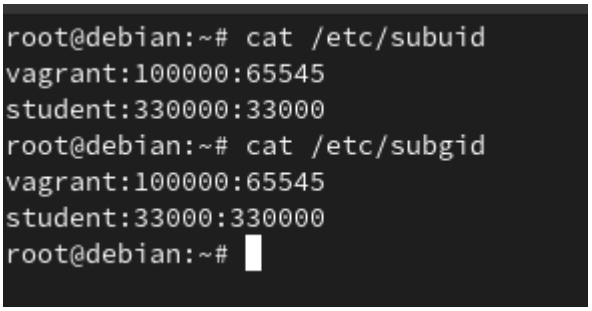
suivi de systemctl restart docker

:



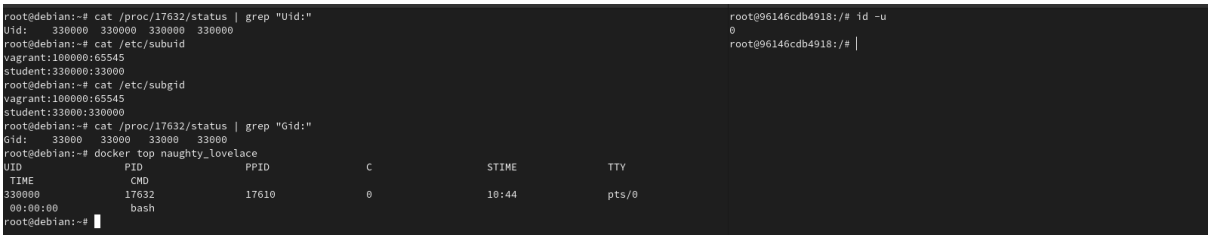
On va pouvoir ainsi mapper les uid/gid des users dans les containers à partir des fichiers /etc/subuid
et /etc/subgid. Modifiez ce fichier.

subuid et subgid



2. Lancez un container avec un processus bash et vérifiez que dans le container ce processus est vu comme appartenant à root et comme appartenant à un utilisateur mappé dans l'hôte.
Pour la suite du TP désactiver les usernamespaces ils sont parfois contraignants quand il faut accéder aux partages sur l'hôte.

le root correspond à l'UId : 0



sudo sysctl kernel.unprivileged_userns_clone=1

permet de désactiver les user namespaces

3.3 Contrôle des ressources allouées aux processus d'un container
3.1 Contrôle des ressources des containers au travers des cgroups
Un attaquant peut saturer un container en lançant un déni de services. Si on ne limite pas les ressources consommées par un container, c'est l'hôte qui sera en manque de ressource à son tour. Il est donc important de limiter les ressources prises par un container via les CGROUPS.

1. A partir de ce Dockerfile.

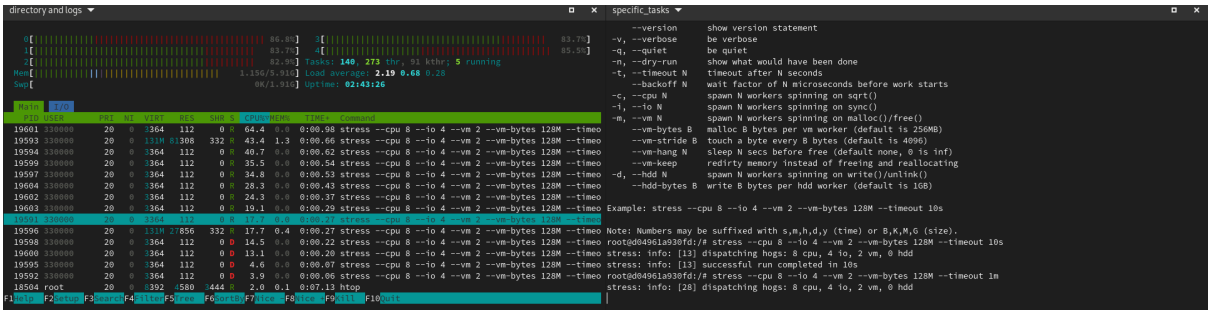
FROM debian:latest
RUN apt-get update && \
apt-get install stress

Générez une image d'un container "stresseur":

Dockerfile.stress

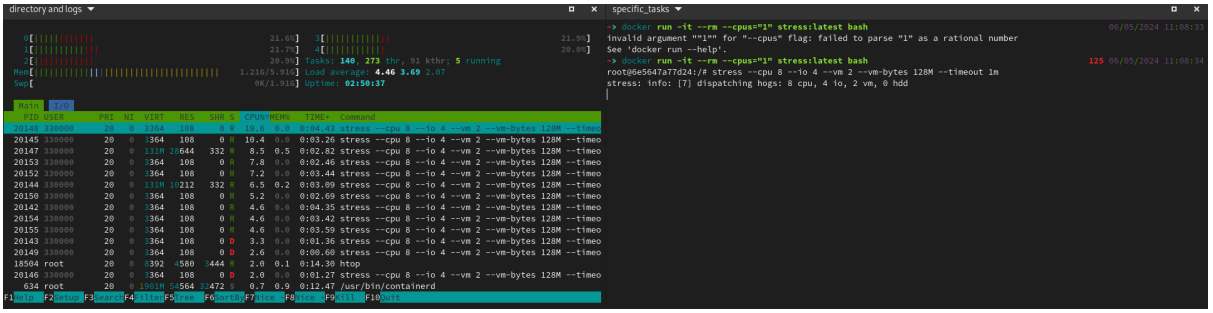
docker build -t stress -f Dockerfile.stress
docker run -it --rm stress:latest bash
stress --cpu 8 --io 4 --vm 2 --vm-bytes 128M --timeout 1m

2. Lancez le container "stresseur" et ouvrez une fenêtre htop pour voir la consommation de ressources
sur l'hôte:



3. Récréez le container et limitez-le à l'utilisation d'un seul CPU.

docker run -it --rm --cpus="1" stress:latest bash
stress --cpu 8 --io 4 --vm 2 --vm-bytes 128M --timeout 1m



3.3.2 Lutte contre l'épuisement des ressources du container et de l'hôte par déni de service
local ("fork bomb" par exemple).

Une forkbomb crée des processus qui vont eux mêmes générer (appel système FORK) d'autres processus fils identiques au père.
La commande suivante permet de lister les ressources du container à l'aide de la commande docker stats:

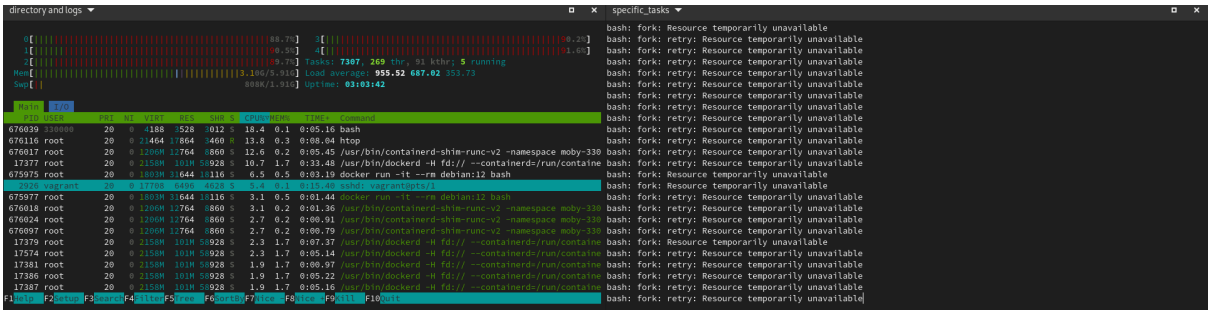
docker stats --no-stream=True

CONTAINER CPU % MEM USAGE / LIMIT MEM % NET I/O BLOCKI/O PIDS

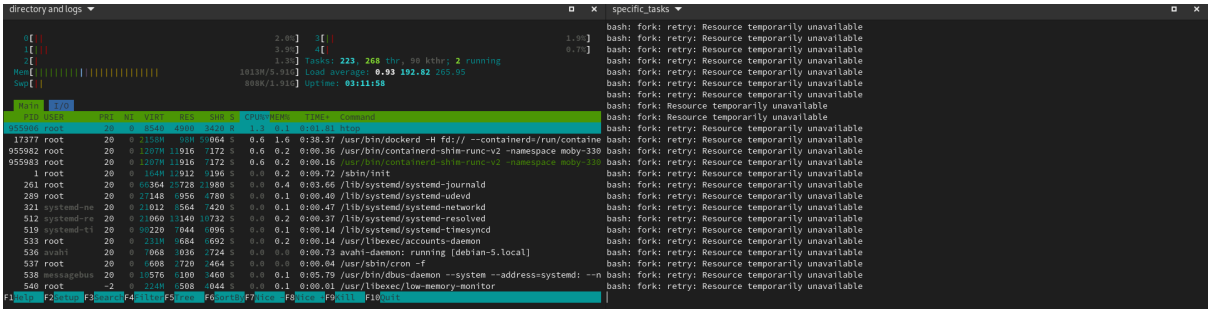
1. Dans votre machine virtuelle Ubuntu lancez une "forkbomb" bash dans un container.

Si tout se passe bien votre container et votre machine virtuelle ne répondront plus, vous voilà
prévenu...
La commande suivante permet de lancer un container contenant une fork bomb:

docker exec -it deb1 /bin/bash -c ":(){ :|:& };:"



2. Trouvez le moyen lors de sa création (docker run) de limiter le nombre de processus dans le container.
docker run --rm -it --pids-limit 100 debian:latest bash



4 Sécurisation des capacités données à un container Docker

4.1 Création d'un container privilégié

Utilisez l'option –privileged lors de la création d'un container. Dans le container et l'aide de la commande capsh et de la commande pscap 2 obtenez les capabilities de votre container et de votre processus bash.

Création du container privilegié :

`docker run --rm -it --privileged debian:12 bash`

```
test@Ndeye-debian:~$ sudo docker run --rm -it --privileged debian:12 bash
[sudo] Mot de passe de test :
docker: Error response from daemon: privileged mode is incompatible with user namespace
s.  You must run the container in the host namespace when running privileged mode.
See 'docker run --help'.
test@Ndeye-debian:~$ nano /etc/docker/daemon.json
```

Utilisation de capsh pour afficher les capabilities de notre container et de notre processus bash :

NB : Il faut utiliser les commandes ci-dessous pour pouvoir l'utiliser :
```
apt-get update
apt-get install libcap2-bin
```

```
root@c1ece4e3dd3c:/# capsh --print
Current: =ep
Bounding set =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
_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_ni
ce,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,cap_audit_read,cap_perfmon,cap_bpf,cap_checkpoint_restore
Ambient set =
Current IAB:
Securebits: 00/0x0/1'b0 (no-new-privs=0)
 secure-noroot: no (unlocked)
 secure-no-suid-fixup: no (unlocked)
 secure-keep-caps: no (unlocked)
 secure-no-ambient-raise: no (unlocked)
uid=0(root) euid=0(root)
gid=0(root)
groups=0(root)
Guessed mode: HYBRID (4)
```

Comparez avec un container non privilegié.

Création du container non-privilegié :

```
test@Ndeye-debian:~$ docker run --rm -it debian:12 bash
root@780349ecc5e6:/# apt-get update
```

Utilisation de capsh pour afficher les capabilities de notre container et de notre processus bash :

```
root@780349ecc5e6:/# capsh --print
Current: cap_chown,cap_dac_override,cap_fowner,cap_fsetid,cap_kill,cap_setgid,cap_setui
d,cap_setpcap,cap_net_bind_service,cap_net_raw,cap_sys_chroot,cap_mknod,cap_audit_write
,cap_setfcap=ep
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_
write,cap_setfcap
Ambient set =
Current IAB: !cap_dac_read_search,!cap_linux_immutable,!cap_net_broadcast,!cap_net_admi
n,!cap_ipc_lock,!cap_ipc_owner,!cap_sys_module,!cap_sys_rawio,!cap_sys_ptrace,!cap_sys_
pacct,!cap_sys_admin,!cap_sys_boot,!cap_sys_nice,!cap_sys_resource,!cap_sys_time,!cap_s
ys_tty_config,!cap_lease,!cap_audit_control,!cap_mac_override,!cap_mac_admin,!cap_syslo
g,!cap_wake_alarm,!cap_block_suspend,!cap_audit_read,!cap_perfmon,!cap_bpf,!cap_checkpo
int_restore
Securebits: 00/0x0/1'b0 (no-new-privs=0)
 secure-noroot: no (unlocked)
 secure-no-suid-fixup: no (unlocked)
 secure-keep-caps: no (unlocked)
 secure-no-ambient-raise: no (unlocked)
uid=0(root) euid=0(root)
gid=0(root)
groups=0(root)
Guessed mode: HYBRID (4)
```

Analyse des Capabilities :

- Container Privilégié:

    - Current: Le champ Current est réduit à =ep, ce qui signifie que le processus a effectivement toutes les capacités possibles dans son ensemble de capacités permises, mais pas nécessairement dans son ensemble de capacités effectives à ce moment précis.
    - Bounding set: Ce champ contient toutes les capacités Linux disponibles, indiquant que le container a le potentiel d'utiliser n'importe quelle capacité système.

- Container Non Privilégié:

    - Current: Les capacités actuelles sont nettement limitées par rapport à un container privilégié. Elles sont principalement concentrées autour de la gestion des fichiers, la modification des UID/GID, et certaines opérations réseau basiques.
    - Bounding set: Le bounding set dans le container non privilégié correspond aux capacités listées sous Current, ce qui signifie qu'il ne peut pas obtenir plus de capacités que celles actuellement assignées.

A quelles capabilities l'option –privileged donne-t-elle accès ?

L'option --privileged donne donc accès à un ensemble complet de capacités Linux, permettant à des applications spécifiques nécessitant un accès étendu au système de fonctionner dans un container.

5 Attaque sur le daemon Docker par un utilisateur local à l'hôte.

1. Sous le compte d'un utilisateur appartenant au groupe Docker, trouvez le moyen d'accéder à /etc/password et /usr/sbin/ de la machine en utilisant les volumes Docker.

On creer d'abord un volume :
docker volume create pswdsbin

puis on lance le lancement du docker avec le volume :
docker run -it -v pswdsbin:/mnt/host -v /etc/mnt/host/etc -v /usr/sbin:/mnt/host/usr/sbin debian:latest bash

2. Qu'en deduisez-vous de la sécurité d'un PC de développeur avec Docker ?

La sécurité au niveau d'un PC de développeur avec Docker est peu fiable, celui-ci provoque alors la possibilité les données sensibles de l'hôte.

NB: les développeurs 🤓 (ps : c'est une petite blague)

6 Utilisation de AppArmor afin de contrôler un container vulnérable à ShellShock

Shellshock est une faille du shell permettant à un attaquant de prendre la main sur une machine. voir:
—
https://www.symantec.com/connect/blogs/shellshock-all-you-need-know-about-bash-bug-vulnerability
— http://www.cert.ssi.gouv.fr/site/CERTFR-2014-ALE-006/index.html

1. Créez le fichier simple-cgi-bin.sh
#!/bin/bash
echo "Content-type: text/plain"
echo
echo
echo "shellshockme if youcan"

2. Générez l'image Docker suivante vulnérable à ShellShock au travers de ce Dockerfile:
Créez un container shell-shock à partir de l'image registry.iutbeziers.fr/debian-lenny-shellshock.
docker pull registry.iutbeziers.fr/debian-lenny-shellshock:latest

docker run -d -p 80:80 --name=hitme --hostname=hitme jmp/shellshockme

3. Vérifiez qu'il est bien vulnérable à ShellShock en lancant un shell dans le container.

```
hitme:/# env val='() { :;}; echo Unexpected command' bash -c "ls"
Unexpected command
bin  boot  dev  etc  home  lib  media  mnt  opt  proc  root  sbin  selinux  srv  sys  tmp  usr  var
hitme:/#
```

4. Vérifiez à l'aide d'un wget que vous pouvez récupérer "à distance" le fichier /etc/password du container.

Pas fonctionnelle avec :
wget -U '() { :;}; /bin/bash -c "Unexpected command"'
http://172.17.0.2:80/simple-cgi-bin.sh

et tous les dérivés

https://www.yeahhub.com/shellshock-vulnerability-exploitation-http-request/

5. Utilisez AppArmor pour empêcher l'exploitation de ce container vulnérable.
Docker charge une configuration apparmor
https://gist.github.com/pushou/3a8a08520ee9895bc9703114ad9c165
Rajoutez y la ligne
deny /usr/lib/cgi-bin/ rwklx, afin de bloquer l'exécution des scripts cgi et donnez ce profile au**
container shellshock lors du run

7 Utilisation de SecComp pour limiter l'utilisation de certains appels systèmes

Seccomp permet de filtrer des appels systèmes en KernelLand. Vous devez vérifier que votre kernel supporte cette fonctionnalité:

cat /boot/config-'uname -r' | grep CONFIG_SECCOMP= CONFIG_SECCOMP=y

et que votre version de SecComp est supérieure à >= 2.2.1. La fonctionnalité a uniquement été
testée avec succès sur Ubuntu 16.

1. Téléchargez le fichier default-no-chmod.json qui va interdire le lancement de la commande chmod
dans le container.
2. Utilisez ce fichier pour limiter les appels systèmes du container lors du "docker run".
3. Expliquez en le fonctionnement.
4. Créez un container et lancez un chmod 777 sur le fichier /etc/password.

8 Utilisation de SecComp pour limiter l'utilisation de certains appels systèmes

Seccomp permet de filtrer des appels systèmes en KernelLand. Vous devez vérifier que votre kernel supporte cette fonctionnalité:
cat /boot/config-uname -r | grep CONFIG_SECCOMP= CONFIG_SECCOMP=y
et que votre version de SecComp est supérieure à >= 2.2.1. La fonctionnalité a uniquement été testée avec succès sur Ubuntu 16.

Vérification du support Seccomp :

```
test@Ndeye-debian:~$ cat /boot/config-$(uname -r) | grep CONFIG_SECCOMP=
CONFIG_SECCOMP=y
test@Ndeye-debian:~$
```

Le fait que la commande retourne CONFIG_SECCOMP=y signifie que Seccomp est activé dans mon noyau.

Création du fichier de politique Seccomp :

On créer un fichier de politique Seccomp nommé chmod.json qui interdira l'exécution de la commande chmod dans le container tel que :

```
 GNU nano 7.2                          chmod.json *
{
  "defaultAction": "SCMP_ACT_ALLOW",
  "syscalls": [
    {
      "name": "chmod",
      "action": "SCMP_ACT_ERRNO"
    }
  ]
}
```

Comme expliqué dans le TP, le fichier configure Seccomp pour permettre tous les appels système sauf chmod, qui sera bloqué (l'appel retournera une erreur).

Lancement du container avec la politique Seccomp

On s'assure que lance un container Docker avec la politique Seccomp appliquée :

`docker run --rm -it --security-opt seccomp=/home/test/chmod.json debian:12 bash`

```
test@Ndeye-debian:~$ docker run --rm -it --security-opt seccomp=/home/test/chmod.json d
ebian:12 bash
root@dcd51058a84a:/#
```

Puis dans le container, on essaie de changer les permissions d'un fichier avec `chmod 777 /etc/passwd` :

```
root@dcd51058a84a:/# chmod 777 /etc/passwd
root@dcd51058a84a:/#
```

Pour une raison que j'ignore, j'arrive à modifier les permissions du fichier à l'intérieur du container alors que la politique Seccomp devrait faire en sorte de retourner une erreur à l'exécution de cette commande si elle est correctement appliquée. Je me suis pourtant bien assuré que le chemin du fichier chmod.json était correct (voir ci-dessous) et que le noyau de mon système prenait bien en charge Seccomp (voir ci-dessus).

```
test@Ndeye-debian:~$ nano chmod.json
test@Ndeye-debian:~$ ls
app.py      docker-compose.yml                      Images   Téléchargements
Bureau      Dockerfile                              Modèles  tpdocker
certs       Documents                               Musique  traefik_dynamic.toml
chmod.json  elastic-agent-8.9.0-linux-x86_64        netflow  Vagrant
dns_sae     elastic-agent-8.9.0-linux-x86_64.tar.gz Public   Vidéos
test@Ndeye-debian:~$ pwd
/home/test
```

J'ai aussi fait en sorte que les permissions du fichier permettent bel et bien à Docker de le lire via `sudo chmod 644 /home/test/chmod.json` qui garantit que le fichier est lisible par tous les utilisateurs (voir ci-dessous).

```
test@Ndeye-debian:~$ sudo chmod 644 /home/test/chmod.json
[sudo] Mot de passe de test :
test@Ndeye-debian:~$
```

9 Containers en lecture seule

Créez un container Web et mettez-le en lecture seule. C'est trivial et idéal pour des containers mettant
à disposition des contenus statiques. Refaites-le mais avec /tmp en lecture-écriture.

Pour cet exemple, nous utiliserons l'image de base nginx, un serveur web populaire, qui sert bien pour distribuer du contenu statique.

Lancement d'un container Nginx en mode lecture seule :

```
test@Ndeye-debian:~$ docker run --rm -it --name web-readonly -p 8080:80 --read-only ngi
nx
/docker-entrypoint.sh: /docker-entrypoint.d/ is not empty, will attempt to perform conf
iguration
/docker-entrypoint.sh: Looking for shell scripts in /docker-entrypoint.d/
/docker-entrypoint.sh: Launching /docker-entrypoint.d/10-listen-on-ipv6-by-default.sh
10-listen-on-ipv6-by-default.sh: info: can not modify /etc/nginx/conf.d/default.conf (r
ead-only file system?)
/docker-entrypoint.sh: Sourcing /docker-entrypoint.d/15-local-resolvers.envsh
/docker-entrypoint.sh: Launching /docker-entrypoint.d/20-envsubst-on-templates.sh
/docker-entrypoint.sh: Launching /docker-entrypoint.d/30-tune-worker-processes.sh
/docker-entrypoint.sh: Configuration complete; ready for start up
2024/05/06 14:09:35 [emerg] 1#1: mkdir() "/var/cache/nginx/client_temp" failed (30: Rea
d-only file system)
nginx: [emerg] mkdir() "/var/cache/nginx/client_temp" failed (30: Read-only file system
)
```

Malheureusement, lors du lancement du container avec l'option --read-only, Nginx a rencontré des erreurs car il tentait de créer et d'écrire dans des répertoires comme /var/cache/nginx, ce qui est impossible sur un système de fichiers en lecture seule.

En analysant les logs du container après son exécution, il est apparu clairement que Nginx nécessitait un accès en écriture à certains répertoires pour fonctionner correctement. Ces

répertoires comprenaient /var/cache/nginx pour le stockage de fichiers temporaires, /var/run pour les fichiers PID, et /var/log/nginx pour les fichiers de logs.

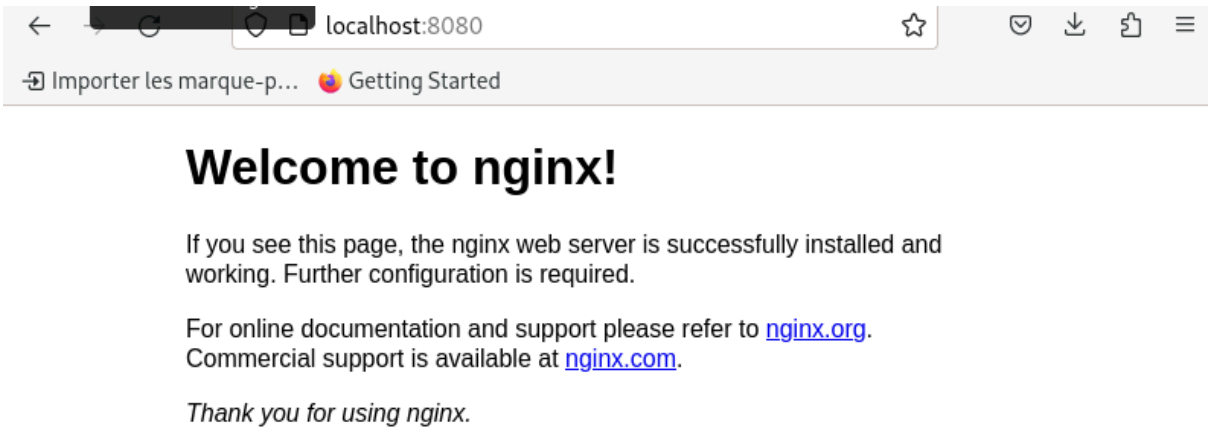
Pour résoudre ce problème, nous avons modifié la commande de lancement du container pour inclure des montages tmpfs pour ces répertoires spécifiques. L'option --tmpfs crée un système de fichiers temporaire en mémoire pour chaque répertoire spécifié, permettant ainsi des écritures temporaires sans compromettre la politique de lecture seule du reste du système de fichiers du container.

La commande finale utilisée pour lancer le container était :

```
docker run --rm -it --name web-readonly -p 8080:80 --read-only \
  --tmpfs /var/cache/nginx \
  --tmpfs /var/run \
  --tmpfs /var/log/nginx \
  nginx
```

Cette configuration a permis à Nginx de démarrer et de fonctionner correctement en mode lecture seule, tout en maintenant les exigences de sécurité pour un environnement de production sécurisé. Le container a pu démarrer sans erreurs et la page web par défaut de Nginx était accessible via le navigateur, confirmant le succès de cette approche :

```
test@Ndeye-debian:~$ docker run --rm -it --name web-readonly -p 8080:80 --read-only \
  --tmpfs /var/cache/nginx \
  --tmpfs /var/run \
  --tmpfs /var/log/nginx \
  nginx
/docker-entrypoint.sh: /docker-entrypoint.d/ is not empty, will attempt to perform conf
iguration
/docker-entrypoint.sh: Looking for shell scripts in /docker-entrypoint.d/
/docker-entrypoint.sh: Launching /docker-entrypoint.d/10-listen-on-ipv6-by-default.sh
10-listen-on-ipv6-by-default.sh: info: can not modify /etc/nginx/conf.d/default.conf (r
ead-only file system?)
/docker-entrypoint.sh: Sourcing /docker-entrypoint.d/15-local-resolvers.envsh
/docker-entrypoint.sh: Launching /docker-entrypoint.d/20-envsubst-on-templates.sh
/docker-entrypoint.sh: Launching /docker-entrypoint.d/30-tune-worker-processes.sh
/docker-entrypoint.sh: Configuration complete; ready for start up
```



10 Contrôle de l'élévation de privilèges dans un container via l'option "-security-opt=no-new-privileges"

Buildez l'image en suivant le README du repo git suivant: https://github.com/pushou/docker-secu-priv
Testez et expliquez l'effet de cette option.

Pour builder l'image en suivant le README du repo git, nous avons donc créé le Dockerfile fourni qui configure un environnement basé sur Debian, ajoute un utilisateur non privilégié, installe un script appelé testnnp dans /tmp, et lui attribue les permissions nécessaires pour être exécuté :

```
  GNU nano 7.2                          Dockerfile
FROM debian:latest
ARG USER=test
ARG GROUP=test
ARG UID=1010
ARG GID=1010
ARG PW=test
RUN groupadd -g ${GID} ${GROUP} && \
    useradd -m -u ${UID} -g ${GID} ${USER} && \
    echo "${USER}:${PW}" | chpasswd
ADD testnnp /tmp/testnnp
RUN chmod +s /tmp/testnnp
USER ${USER}
CMD ["/tmp/testnnp"]
```

On a donc ensuite créé un script testnnp, qui ici est conçu pour afficher l'UID effectif de l'utilisateur qui l'exécute (comme l'exemple du git), permettant ainsi de tester les impacts de l'option de sécurité --security-opt=no-new-privileges :

```
  GNU nano 7.2                          testnnp
#!/bin/bash
echo "Effective UID: $(id -u)"
```

```
test@Ndeye-debian:~$ docker build -t testnnp .
[+] Building 1.2s (9/9) FINISHED                                    docker:default
 => [internal] load build definition from Dockerfile                          0.0s
 => => transferring dockerfile: 331B                                          0.0s
 => [internal] load metadata for docker.io/library/debian:latest              0.7s
 => [internal] load .dockerignore                                             0.0s
 => => transferring context: 2B                                               0.0s
 => [1/4] FROM docker.io/library/debian:latest@sha256:1aadfee8d292f64b045adb830f  0.0s
 => [internal] load build context                                             0.0s
 => => transferring context: 123B                                             0.0s
 => CACHED [2/4] RUN groupadd -g 1010 test &&     useradd -m -u 1010 -g 1010 tes  0.0s
 => [3/4] ADD testnnp /tmp/testnnp                                            0.1s
 => [4/4] RUN chmod +s /tmp/testnnp                                           0.3s
 => exporting to image                                                        0.1s
 => => exporting layers                                                       0.0s
 => => writing image sha256:caa356cf994162e6abe42e8a777c4dc7539e3dda76e53943f7d2  0.0s
 => => naming to docker.io/library/testnnp                                    0.0s
```

Après cela, nous avons pu exécuter le container sans et avec l'option --security-opt=no-new-privileges :

```
test@Ndeye-debian:~$ docker run -it --rm testnnp
Effective UID: 1010
test@Ndeye-debian:~$ docker run -it --rm --security-opt=no-new-privileges testnnp
Effective UID: 1010
```

Lors de l'exécution sans l'option, le script a correctement affiché l'UID de l'utilisateur non privilégié (1010), et de manière identique avec l'option activée.

Je trouvais ce résultat étrange étant donné le même test sur le git ou les 2 résultats diffèrent :

> ## build and run with && without
>
> ```
> docker build -t testnnp .
> docker run -it --rm  testnnp
> Effective uid: 0
>
> docker run -it --rm  --security-opt=no-new-privileges  testnnp
> Effective uid: 1000
> ```

Comme pour l'exemple du git ci-dessus, notre script se contente d'afficher l'UID de l'utilisateur actuel sans tenter d'exploiter le bit setuid pour effectuer des opérations qui nécessitent des privilèges élevés. Ainsi, puisque le script ne met pas en œuvre de telles actions, l'effet de l'option no-new-privileges n'est pas visible dans notre test.
Pour observer un comportement similaire à celui décrit dans le git, il aurait fallu que le script testnnp inclue des commandes qui tentent explicitement d'utiliser des privilèges élevés,

comme modifier des fichiers appartenant à root ou changer des configurations système critiques. Faute de temps nous ne l'avons pas fait.

11 Kata containers
Il s'agit de faire tourner un container dans une machine virtuelle. La solution des "kata containers" permet de faire exécuter un container dans une machine virtuele KVM ou FireCracker (VMS d'AWS). On va tester ici la solution avec KVM:

sudo snap install kata-containers --classic

Modifiez le fichier daemon.json et redémarrer Docker

```
{
"insecure-registries" : ["registry.infres.local"],
"default-runtime": "runc",
"runtimes": {
kata-runtime": {
"path": "/snap/bin/kata-containers.runtime"
}
}
}
```