SSE - Conteneurs

TP8

Master 2 Cybersécurité

2018 - 2019

Encadré par :
Josselin MARIETTE

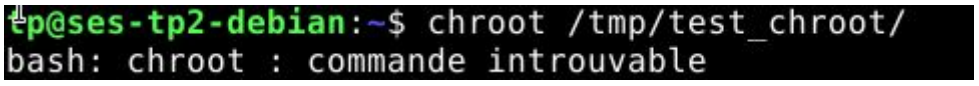
Réalisé par :
Manon DEROCLES
Alexis LE MASLE

Table Des Matières

Mécanismes techniques

Exercice 3-1

Pourquoi est-ce que le premier appel à la commande chroot (celui sans sudo) ne fonctionne pas? La deuxième appel fonctionne ? Pourquoi ?



Appel à la commande chroot sans le sudo

Lors de la première commande nous ne pouvons pas accéder à la commande chroot.
En effet, cette commande est un appel système permettant pour un processus donné de changer de répertoire racine.

Appel à la commande chroot avec le sudo

Lors de la deuxième commande, chroot fonctionne. L'erreur vient de /bin/bash et non de chroot. En effet, la commande chroot est exécuté avec les droits root.
La commande ne se réalise pas car il nous manque des dépendances, en effet avec chroot nous créons notre propre "système", un cloisonnement propre. Nous n'avons donc plus les dépendances qui se trouvent dans le système à l'emplacement racine de la machine hôte, il faut les mettre également dans la nouvelle racine créée par chroot.

Quelle différence peut-on faire entre l'appel à exit et l'utilisation du programme unchroot ? Quels sont les privilèges nécessaires pour faire fonctionner le programme unchroot ?
Avec exit, nous arrêtons / tuons le processus courant. Avec unchroot, nous créons une nouvelle prison et nous définissons la nouvelle racine étant celle de la racine de la machine hôte. Lors d'un unchroot nous créons un nouveau dossier à la racine de la prison. Puis nous créons une nouvelle prison dans l'ancienne prison. Mais le programme unchroot a gardé le pointeur du nouveau dossier créé qui est en dehors de l'arborescence de cette nouvelle prison, nous nous en servons pour remonter jusqu'à la racine de la machine hôte. Lors que nous sommes sur la racine de la machine hôte nous créons une nouvelle prison, mais celle ci n'a aucune restriction puisqu'elle a accès à tout le système.

```c
#include <sys/stat.h>
#include <unistd.h>
#include <fcntl.h>
#include <errno.h>

#define NODES_TO_ROOT   100

int main() {

    mkdir(".escape", 0755);
    chroot(".escape");

    chdir(".");
    for(int i = 0; i < NODES_TO_ROOT; i++)
        chdir("..");
    chroot(".");

    return execl("/bin/sh", "-i", NULL);
}
```

code du fichier unchroot.c

Au final, la commande chroot permet-elle une bonne isolation au niveau système de fichiers ? Que pourrait-on envisager de faire pour l'améliorer ?

La commande chroot crée une prison, une nouvelle arborescence de fichier. Cette commande permet un bon cloisonnement seulement si elle ne peut pas s'échapper et le seul moyen de s'en évader et de posséder un pointeur qui est externe à l'arborescence du fichier de la prison.
Une prison est très utile car elle permet de limiter l'accès à des fichier sensible en dehors de la prison si celle ci est compromise. Et en cas de compromission l'attaquant ne peut pas sortir de la prison.

Pour améliorer le chroot, il faudrait mettre des restrictions par exemple ne pas pouvoir créer des bash…

Exercice 3-2

Que remarquez-vous concernant les processus et leur PID ? Que pouvez-vous conclure sur le rôle du namespace PID ?

Nous pouvons voir sur la capture d'écran que le PID du processus top n'est pas le même entre la machine hôte et le bash que nous avons créé. Nous constatons que nous avons cloisonné le processus top à notre bash.

```
top - 17:13:01 up  1:01,  1 user,  load average: 0,00, 0,00, 0,00
Tasks:   2 total,   1 running,   1 sleeping,   0 stopped,   0 zombie
%Cpu(s):  0,0 us,  0,0 sy,  0,0 ni,100,0 id,  0,0 wa,  0,0 hi,  0,0 si,  0,0 st
KiB Mem :  1020332 total,   676180 free,   158108 used,   186044 buff/cache
KiB Swap:  1044476 total,  1044476 free,        0 used.   723692 avail Mem

  PID USER      PR  NI    VIRT    RES    SHR S %CPU %MEM     TIME+ COMMAND
    1 root      20   0   19872   3704   3148 S  0,0  0,4   0:00.00 bash
    3 root      20   0   44772   3588   3104 R  0,0  0,4   0:00.00 top
```

```
Terminal - tp@ses-tp2-debian: ~
Fichier  Édition  Affichage  Terminal  Onglets  Aide
tp@ses-tp2-debian:~$ ps aux | grep top
tp         619  0.0  3.7 488760 38100 ?        Sl   16:11   0:00 xfdesktop --display :0.0 --sm-clien
t-id 20fbd079f-72d3-4140-a095-bd5977232531
root       777  0.0  0.3  44772  3680 pts/0    S+   16:22   0:01 top
tp         840  0.0  0.0  12784   944 pts/1    S+   17:11   0:00 grep top
```

Les deux terminaux affichant la commande top

Que constatez-vous en examinant les différences ?

```
root@ses-tp2-debian:/home/tp# ls -l /proc/$$/ns
total 0
lrwxrwxrwx 1 root root 0 janv. 18 17:33 cgroup -> cgroup:[4026531835]
lrwxrwxrwx 1 root root 0 janv. 18 17:33 ipc -> ipc:[4026531839]
lrwxrwxrwx 1 root root 0 janv. 18 17:33 mnt -> mnt:[4026532182]
lrwxrwxrwx 1 root root 0 janv. 18 17:33 net -> net:[4026531957]
lrwxrwxrwx 1 root root 0 janv. 18 17:33 pid -> pid:[4026532183]
lrwxrwxrwx 1 root root 0 janv. 18 17:33 user -> user:[4026531837]
lrwxrwxrwx 1 root root 0 janv. 18 17:33 uts -> uts:[4026531838]
```

Le terminal sur le bash forké

```
sudo ls -l /proc/777/ns
```

```
[sudo] Mot de passe de tp :
total 0
lrwxrwxrwx 1 root root 0 janv. 18 17:42 cgroup -> cgroup:[402653183
lrwxrwxrwx 1 root root 0 janv. 18 17:42 ipc -> ipc:[4026531839]
lrwxrwxrwx 1 root root 0 janv. 18 17:42 mnt -> mnt:[4026531840]
lrwxrwxrwx 1 root root 0 janv. 18 17:42 net -> net:[4026531957]
lrwxrwxrwx 1 root root 0 janv. 18 17:42 pid -> pid:[4026531836]
lrwxrwxrwx 1 root root 0 janv. 18 17:42 user -> user:[4026531837]
lrwxrwxrwx 1 root root 0 janv. 18 17:42 uts -> uts:[4026531838]
tp@ses-tp2-debian:~$ sudo ls -l /proc/$$/ns
total 0
lrwxrwxrwx 1 tp tp 0 janv. 18 17:29 cgroup -> cgroup:[4026531835]
lrwxrwxrwx 1 tp tp 0 janv. 18 17:29 ipc -> ipc:[4026531839]
lrwxrwxrwx 1 tp tp 0 janv. 18 17:29 mnt -> mnt:[4026531840]
lrwxrwxrwx 1 tp tp 0 janv. 18 17:29 net -> net:[4026531957]
lrwxrwxrwx 1 tp tp 0 janv. 18 17:29 pid -> pid:[4026531836]
lrwxrwxrwx 1 tp tp 0 janv. 18 17:29 user -> user:[4026531837]
lrwxrwxrwx 1 tp tp 0 janv. 18 17:29 uts -> uts:[4026531838]
```

Le terminal sur la machine hôte

Le pid n'est pas le même ainsi que le mnt. Cela est normal car lors de la création du bash nous créons un nouveau processus en séparant les namespaces entre le parent et le fils mount (sudo unshare **--fork --pid --mount-proc** bash).

Quel résultat obtenez-vous ?

```
tp@ses-tp2-debian:~$ sudo unshare --fork --pid bash
[sudo] Mot de passe de tp :
root@ses-tp2-debian:/home/tp# echo $$
1
root@ses-tp2-debian:/home/tp# ps -up 1
USER       PID %CPU %MEM    VSZ   RSS TTY      STAT START   TIME COMMAND
root         1  0.0  0.6  57064  6888 ?        Ss   16:11   0:00 /sbin/ini
root@ses-tp2-debian:/home/tp# ls -l /proc/$$/n
net/       ns/        numa_maps
root@ses-tp2-debian:/home/tp# ls -l /proc/$$/ns
total 0
lrwxrwxrwx 1 root root 0 janv. 18 17:50 cgroup -> cgroup:[4026531835]
lrwxrwxrwx 1 root root 0 janv. 18 17:50 ipc -> ipc:[4026531839]
lrwxrwxrwx 1 root root 0 janv. 18 17:50 mnt -> mnt:[4026531840]
lrwxrwxrwx 1 root root 0 janv. 18 17:50 net -> net:[4026531957]
lrwxrwxrwx 1 root root 0 janv. 18 17:50 pid -> pid:[4026531836]
lrwxrwxrwx 1 root root 0 janv. 18 17:50 user -> user:[4026531837]
lrwxrwxrwx 1 root root 0 janv. 18 17:50 uts -> uts:[4026531838]
```

Le terminal du bash forké

La valeur de mnt n'est plus différente.

Pouvez-vous émettre une hypothèse qui explique la différence ?

```
tp@ses-tp2-debian:~$ sudo unshare --fork --mount-proc --pid bash
[sudo] Mot de passe de tp :
root@ses-tp2-debian:/home/tp# ls -l /proc/ | wc -l
60
```

Terminal - tp@ses-tp2-debian: ~

Fichier Édition Affichage Terminal Onglets Aide

```
tp@ses-tp2-debian:~$ sudo unshare --fork --pid bash
[sudo] Mot de passe de tp :
root@ses-tp2-debian:/home/tp# ls -l /proc/ | wc -l
181
```

Les terminaux affichant le nombre de pid présent dans les bash

L'option --mount-proc modifie le répertoire /proc. Cette option permet de créer également un nouvel espace de noms pour les mount car sinon cela entraînerait des erreurs avec les programmes déjà existant sur le système. Le nouveau répertoire est monté en privé.

Sans cette option d'activée nous sommes toujours dans le répertoire du système hôte /proc. Et avec cette option nous sommes dans un répertoire propre à notre bash monté.

Exercice 3-3

Afficher les différents types de contrôleurs de ressources (man cgroups pour une description du rôle de chaque contrôleur) présents sur le système, puis ceux qui

impactent le shell courant. Le shell courant est-il suivi par tous les contrôleurs présents ? Pour chaque contrôleur, est-il rattaché à un cgroup mis en place par SystemD ou le cgroup par défaut (cgroup qui n'a pas de nom) ?

La commande cgroups permettent d'isoler plusieurs fonctionnalités, comme une limitation sur la mémoire, le cache, les ressources processeur, la priorité des entrées sorties, masquer les processus, les connexions réseaux, les fichiers, et permet de sauvegarder à un instant T, le mettre en pause ou le redémarrer à ce point T.

```
tp@ses-tp2-debian:~$ mount -t cgroup
cgroup on /sys/fs/cgroup/systemd type cgroup (rw,nosuid,nodev,noexec,relatime,xattr,release_agent=/l
ib/systemd/systemd-cgroups-agent,name=systemd)
cgroup on /sys/fs/cgroup/cpu,cpuacct type cgroup (rw,nosuid,nodev,noexec,relatime,cpu,cpuacct)
cgroup on /sys/fs/cgroup/freezer type cgroup (rw,nosuid,nodev,noexec,relatime,freezer)
cgroup on /sys/fs/cgroup/perf_event type cgroup (rw,nosuid,nodev,noexec,relatime,perf_event)
cgroup on /sys/fs/cgroup/blkio type cgroup (rw,nosuid,nodev,noexec,relatime,blkio)
cgroup on /sys/fs/cgroup/devices type cgroup (rw,nosuid,nodev,noexec,relatime,devices)
cgroup on /sys/fs/cgroup/net_cls,net_prio type cgroup (rw,nosuid,nodev,noexec,relatime,net_cls,net_p
rio)
cgroup on /sys/fs/cgroup/pids type cgroup (rw,nosuid,nodev,noexec,relatime,pids)
cgroup on /sys/fs/cgroup/cpuset type cgroup (rw,nosuid,nodev,noexec,relatime,cpuset)
cgroup on /sys/fs/cgroup/memory type cgroup (rw,nosuid,nodev,noexec,relatime,memory)
```

Résultat de la commande mount -t cgroup

Les fonctionnalités concernées dans ce shell sont: le système d'initialisation, le cpu, management des tâches système, le contrôleur, les pids et la mémoire.

```
tp@ses-tp2-debian:~$ ls /sys/fs/cgroup/
blkio  cpuacct      cpuset   freezer  net_cls          net_prio    pids
cpu    cpu,cpuacct  devices  memory   net_cls,net_prio  perf_event  systemd
tp@ses-tp2-debian:~$ cat /proc/$$/cgroup
10:memory:/
9:cpuset:/
8:pids:/user.slice/user-1000.slice/session-2.scope
7:net_cls,net_prio:/
6:devices:/user.slice
5:blkio:/
4:perf_event:/
3:freezer:/
2:cpu,cpuacct:/
1:name=systemd:/user.slice/user-1000.slice/session-2.scope
```

Résultat des commandes ls et cat

Pour isoler les propriétés nommées précédemment, cgroups crée des répertoires privé qui contiennent ces propriétés ci.
Pour la mémoire, le cpu, le manager des tâches sont fait par un cgroup de défaut (ils ne possèdent pas de noms propre). Pour les autres ils sont gérés par le SystemD par le nom user.slice.

Combien de PID sont affichés par la dernière commande. Comment l'expliquez-vous ?

```
tp@ses-tp2-debian:~$ sudo su
[sudo] Mot de passe de tp :
root@ses-tp2-debian:/home/tp# cd /sys/fs/cgroup/pids/
root@ses-tp2-debian:/sys/fs/cgroup/pids# mkdir tp
root@ses-tp2-debian:/sys/fs/cgroup/pids# echo "$$" > tp/tasks
root@ses-tp2-debian:/sys/fs/cgroup/pids# grep prids /proc/$$/cgroup
root@ses-tp2-debian:/sys/fs/cgroup/pids# cat tp/tasks
1098
1108
```

Nombre de pid dans le nouveau cgroup tp

Il y a deux pid dans le cgroup tp. Le pid 1098 correspond au processus bash.
Le pid 1108 correspond au dernier processus utilisé donc logiquement il correspond au processus cat.

Quel résultat obtient-on ici ? La valeur contenu dans le fichier pids.event a-t-elle évolué ?

```
root@ses-tp2-debian:/sys/fs/cgroup/pids# cat tp/pids.events
max 0
root@ses-tp2-debian:/sys/fs/cgroup/pids# echo 2 > tp/pids.max
root@ses-tp2-debian:/sys/fs/cgroup/pids# sleep 20s &
[1] 1223
root@ses-tp2-debian:/sys/fs/cgroup/pids# cat tp/pids.current
bash: fork: retry: Ressource temporairement non disponible
bash: fork: retry: Ressource temporairement non disponible
bash: fork: retry: Ressource temporairement non disponible
bash: fork: retry: Ressource temporairement non disponible
bash: fork: Appel système interrompu
[1]+  Fini                    sleep 20s
root@ses-tp2-debian:/sys/fs/cgroup/pids# cat tp/pids.events
max 4
```

Liste des commandes

Le fichier **tp/pids.events** contient bien le chiffre 4, qui correspond aux 4 tentatives durant lesquels la ressource était occupée.

Quel est le comportement de la machine dans les 2 cas ?

Lorsque nous mettons le max à 100, au bout d'un certain temps la commande termine son exécution, nous avons de-nouveau accès à la machine.

Lorsque nous mettons le max à “max”, nous perdons la machine, en effet la machine ne réponds plus puisque nous ne mettons plus de limite à la création de PID.

Quel « fichier » de configuration faut-il utiliser pour mettre en place la limite fixée ?

Docker

Exercice 4-1

A l'aide du processus sleep lancé dans le conteneur, identifier l'arborescence des processus parents du conteneur bash puis noter le PID de chaque processus et le nom de l'utilisateur ayant lancé chaque processus.

Dans un conteneur docker nous lançons la commande:

```
bash-5.0# sleep 60s
```

Sur la machine hôte nous identifions avec la commande *ps -AF*
Nous obtenons alors en suivant les pid et ppid l'arborescence suivante:

```
/sbin/init
containerd
containerd-shim
bash
sleep
```

L'utilisateur associé est "root" pour chacun de ces processus.

A l'aide des commandes courantes (top, free, ps, ...), identifier depuis le conteneur la quantité de mémoire exposée dans le conteneur ainsi que les processus présents et leur PID. Que remarque-t-on ? Est-il possible de formuler une hypothèse sur la gestion des PID ?

```
tp@ses-tp2-debian:~$ free
              total        used        free      shared  buff/cache   available
Mem:        1020332      255932      191808        3624      572592      608572
Swap:       1044476        3012     1041464
```

Capture de la mémoire de l'hôte

```
bash-5.0# free
              total        used        free      shared     buffers      cached
Mem:        1020332      828400      191932        3624       44352      479216
-/+ buffers/cache:      304832      715500
Swap:       1044476        3012     1041464
```

Capture de la mémoire du conteneur

On peut constater que le conteneur docker à accès à l'intégralité de la mémoire. Les pid présent sur le conteneur sont seulement 1 pour bash et le pid de la commande en cours, on peut faire une hypothèse sur le fait que les autres processus sont tous gérés par le système hôte.

```
Mem: 830764K used, 189568K free, 3924K shrd, 45300K buff, 479936K
CPU:   3% usr   0% sys   0% nic  95% idle   0% io   0% irq   0% s
Load average: 0.01 0.03 0.00 3/229 109
  PID  PPID USER     STAT   VSZ %VSZ CPU %CPU COMMAND
    1     0 root     S     4564   0%   0   0% bash
  109     1 root     R     1524   0%   0   0% top
```

Processus courant sur le conteneur

Quels sont les types de namespace mis en place spécifiquement pour le conteneur bash ? D'où proviennent les namespace qui ne sont pas mis en place spécifiquement pour le conteneur ?

```
tp@ses-tp2-debian:~$ sudo lsns
        NS TYPE   NPROCS   PID USER  COMMAND
4026531835 cgroup    134     1 root  /sbin/init
4026531836 pid       133     1 root  /sbin/init
4026531837 user      134     1 root  /sbin/init
4026531838 uts       133     1 root  /sbin/init
4026531839 ipc       133     1 root  /sbin/init
4026531840 mnt       130     1 root  /sbin/init
4026531857 mnt         1    13 root  kdevtmpfs
4026531957 net       132     1 root  /sbin/init
4026532106 mnt         1   309 root  /lib/systemd/systemd-udevd
4026532132 net         1   500 rtkit /usr/lib/rtkit/rtkit-daemon
4026532185 mnt         1   500 rtkit /usr/lib/rtkit/rtkit-daemon
4026532202 mnt         1 10515 root  bash
4026532203 uts         1 10515 root  bash
4026532204 ipc         1 10515 root  bash
4026532205 pid         1 10515 root  bash
4026532207 net         1 10515 root  bash
tp@ses-tp2-debian:~$ sudo ls -l /proc/10515/ns
total 0
lrwxrwxrwx 1 root root 0 janv. 18 17:34 cgroup -> cgroup:[4026531835]
lrwxrwxrwx 1 root root 0 janv. 18 17:02 ipc -> ipc:[4026532204]
lrwxrwxrwx 1 root root 0 janv. 18 17:02 mnt -> mnt:[4026532202]
lrwxrwxrwx 1 root root 0 janv. 18 16:47 net -> net:[4026532207]
lrwxrwxrwx 1 root root 0 janv. 18 17:02 pid -> pid:[4026532205]
lrwxrwxrwx 1 root root 0 janv. 18 17:02 user -> user:[4026531837]
lrwxrwxrwx 1 root root 0 janv. 18 17:02 uts -> uts:[4026532203]
```

En observant les namespaces on constate que les namespaces associé au bash du conteneur Docker (pid 10515) sont associés par lien symbolique aux namespaces de l'hôte. Par exemple cgroup fais un lien symbolique vers le namespace du cgroup de l'hôte ou encore le namespace user vers celui de l'hôte aussi.

Cela confirme-t-il l'hypothèse émise précédemment ? S'agit-il d'une bonne pratique en terme de sécurité ?

Cela confirme l'hypothèse, on se rend compte que les namespaces des conteneurs sont liés à ceux de l'hôte. Il ne s'agit pas d'une bonne pratique car docker a un accès "root" à la machine et une faille sur les conteneurs Docker pourrait avoir des conséquences sur le système hôte directement.

Identifier si docker utilise les cgroups pour le conteneur bash et comparer la vision des cgroup depuis la machine debian et le conteneur bash (remplacer 1114 par le PID du processus bash du conteneur) à partir des informations disponibles pour les processus :

```
tp@ses-tp-debian:~$ cat /proc/1114/cgroup
bash-5.0# cat /proc/$$/cgroup
```

En affichant les “cgroup” du conteneur et de l’hôte, nous obtenons le même résultat:

```
bash-5.0# cat /proc/$$/cgroup
10:cpuset:/docker/8c088a001710dd01b9ea42858a7a7eb7796300d0bb45f987b2e7bc4465c1d5fb
9:memory:/docker/8c088a001710dd01b9ea42858a7a7eb7796300d0bb45f987b2e7bc4465c1d5fb
8:perf_event:/docker/8c088a001710dd01b9ea42858a7a7eb7796300d0bb45f987b2e7bc4465c1d5fb
7:cpu,cpuacct:/docker/8c088a001710dd01b9ea42858a7a7eb7796300d0bb45f987b2e7bc4465c1d5fb
6:freezer:/docker/8c088a001710dd01b9ea42858a7a7eb7796300d0bb45f987b2e7bc4465c1d5fb
5:pids:/docker/8c088a001710dd01b9ea42858a7a7eb7796300d0bb45f987b2e7bc4465c1d5fb
4:net_cls,net_prio:/docker/8c088a001710dd01b9ea42858a7a7eb7796300d0bb45f987b2e7bc4465c1d5fb
3:blkio:/docker/8c088a001710dd01b9ea42858a7a7eb7796300d0bb45f987b2e7bc4465c1d5fb
2:devices:/docker/8c088a001710dd01b9ea42858a7a7eb7796300d0bb45f987b2e7bc4465c1d5fb
1:name=systemd:/docker/8c088a001710dd01b9ea42858a7a7eb7796300d0bb45f987b2e7bc4465c1d5fb
```

On peut déduire que le conteneur utilise le même cgroup que l’hôte. En effet les chemins sont exactement les mêmes pour chacun des cgroups, celui du conteneur bash ou celui de l’hôte.

Comparer la valeur de /sys/fs/cgroup/memory/memory.limit_in_bytes pour les 2 conteneurs, puis la quantité de mémoire indiquée par la commande free. Sont-elles identiques pour les 2 conteneurs ? Que peut-on en conclure ?

```
bash-5.0# cat /sys/fs/cgroup/memory/memory.limit_in_bytes
9223372036854771712
```

Premier conteneur

```
bash-5.0# cat /sys/fs/cgroup/memory/memory.limit_in_bytes
104857600
bash-5.0# free
             total       used       free     shared    buffers
 cached
Mem:       1020332     880304     140028       4204      49676
 501556
-/+ buffers/cache:     329072     691260
Swap:      1044476       2840    1041636
```

Second conteneur

Les deux conteneurs n'affichent pas la même valeur pour "memory.limit_in_bytes" mais la valeur de "free" est la même. Il semblerait que Docker prenne toutes les ressources de la machine hôte par défaut, mais que l'on peut définir des limitations au lancement d'un conteneur comme avec l'option "-m 100m". En définissant des limites lors du lancement d'un conteneur, cgroup est modifié pour cloisonner aux ressources précisées.

Comparer la valeur de /sys/fs/cgroup/pids/pids.max pour les 2 conteneurs. Est-elle identique pour les 2 conteneurs ? Le premier conteneur est-il sensible a un type d'attaque en particulier ?

Le premier conteneur sans limitations a la valeur "max" dans le fichier "/sys/fs/cgroup/pids/pids.max" dans le second conteneur nous obtenons la valeur "3".

Le conteneur ayant un pid max à "max" rend sensible aux attaques de type "bombe fork" en créant une boucle infini de fork jusqu'à saturation du système par manque de ressources et menant au crash.

Exercice 4-2

Comparer et commenter les capacités attribuées au processus sleep du conteneur bash dans les 4 cas suivants. Quelle recommandation pourrait-on émettre sur la gestion des capacités dans les conteneurs ?

```
tp@ses-tp2-debian:~$ sudo docker run --rm -d bash sleep 5s > /dev/null; pscap
| grep sleep
2900  2918  root        sleep             chown, dac_override, fowner, fsetid,
 kill, setgid, setuid, setpcap, net_bind_service, net_raw, sys_chroot, mknod,
audit_write, setfcap
```

```
tp@ses-tp2-debian:~$ sudo docker run --rm -d --privileged bash sle
ep 5s > /dev/null; pscap | grep sleep
3055  3072  root        sleep             full
```

```
tp@ses-tp2-debian:~$ sudo docker run --rm -d --cap-drop=all bash sleep 5s > /dev/null; pscap
ppid  pid   name        command           capabilities
1     275   root        systemd-journal   chown, dac_override, dac_read_search, fowner, setgid, setuid, sys_ptrace, sys_admin, audit_c
ontrol, mac_override, syslog, audit_read
1     291   root        systemd-udevd     full
1     312   root        lvmetad           full
1     498   root        rsyslogd          full
1     500   root        systemd-logind    chown, dac_override, dac_read_search, fowner, kill, sys_admin, sys_tty_config, audit_control
, mac_admin
1     501   root        cron              full
1     502   messagebus  dbus-daemon       audit_write +
1     519   rtkit       rtkit-daemon      dac_read_search, sys_nice
1     538   root        dhclient          full
1     556   root        dockerd           full
1     568   root        containerd        full
1     571   root        agetty            full
556   582   root        containerd        full
1     820   root        lightdm           full
1     831   root        VBoxService       full
820   840   root        Xorg              full
820   891   root        lightdm           full
1     988   root        polkitd           full
1     1008  root        upowerd           full
1     1106  root        udisksd           full
1     1383  root        packagekitd       full
582   3389  root        containerd-shim   full
```

```
tp@ses-tp2-debian:~$ sudo docker run --rm -d --cap-drop=all --cap-
add=net_bind_service bash sleep 5s > /dev/null; pscap | grep sleep
3510  3528  root        sleep             net_bind_service
```

Lors du lancement du conteneur bash, l'option "--privileged" permet d'attribuer tous les privilèges.
"--cap-drop=all" permet de retirer toutes les capacités
"--cap-drop=all --cap-add=net_bind_service" va supprimer toutes les capacités et ajouter seulement "net_bind_service".

Une meilleure pratique serait de bloquer toutes les capacités par défaut et de n'ajouter que les capacités nécessaires au fonctionnement du conteneur. Cela permet de réduire les droits attribués au conteneur et de maîtriser ses permissions.